Universidade de Aveiro Departamento de
Electrónica, Telecomunicações e Informática,
2017

Marco Filipe
Moutinho da Silva

Modular platform for detection of BGP routing attacks

Plataforma modular para deteção de ataques de encaminhamento BGP

Universidade de Aveiro Departamento de
2017 Electrónica, Telecomunicações e Informática,

**Marco Filipe
Moutinho da Silva**

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Paulo Jorge Salvador Serra Ferreira, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president	**Professor Doutor André Ventura da Cruz Marnoto Zúquete** Professor Auxiliar, Universidade de Aveiro
vogais / examiners committee	**Professor Doutor Rui Jorge Morais Tomaz Valadas** Professor Catedrático, Universidade Técnica de Lisboa
	Professor Doutor Paulo Jorge Salvador Serra Ferreira Professor Auxiliar, Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

Em primeiro lugar, agradeço aos meus pais. Por todo o apoio e sacrifício que fizeram durante toda a minha jornada académica. Sem eles nada disto seria possível. É imensurável a gratidão e o amor que sinto por vocês. Ao meu irmão, por me ter ajudado e estado presente quando era necessário.
De seguida, ao meu orientador. Por toda a paciência, ajuda e motivação para realizar este trabalho da melhor forma possível.
Aos meus três grandes amigos e companheiros de viagem: Leandro Ricardo, João Simões e David Silva. A vós um grande abraço, um agradecimento pela amizade e camaradagem ao longo destes anos todos. Espero que se mantenham por perto!
À Inês Moreira, por se ter tornado numa parte tão importante da minha vida, e me ter 'dado na cabeça' quando a vontade era desistir. Obrigado pela tua companhia e genuína amizade.
À Mariana Poço, pelo ombro amigo quando mais foi necessário.
A todos os amigos, amigas e colegas que, de uma maneira ou outra, estiveram comigo nesta aventura. Um abraço.

Resumo

Para a conectividade da Internet ser possível, foram criados protocolos de encaminhamento para esse propósito. O protocolo de encaminhamento global utilizado é o BGP, que utiliza a agregação de vários prefixos de redes em ASes de maneira a criar um grafo que contém a informação sobre as rotas para os diversos prefixos de redes públicas, criando assim as condições para conectividade global. Apesar de satisfazer o seu propósito, este protocolo é baseado em confiança cega entre pares de BGP levando a que este fique exposto a ataques. Sendo este protocolo responsável pela conectividade global, um ataque efetuado através deste protocol pode levar a que o tráfego seja desviado da sua rota normal e vá parar às mãos do atacante, dando a possibilidade de este conseguir ler e ou alterar o seu conteúdo. Apesar de medidas de segurança já terem sido propostas, estas não estão atualmente implementadas na maioria dos dispositivos que utilizam este protocolo, deixando-os assim vulneráveis a dispositivos comprometidos, podendo comprometer as suas tabelas de encaminhamento.

Os ISPs (provedor de serviço de Internet) têm metodologias para detetar este tipo de ataques e agir sobre eles mas os utilizadores, tais como privados e ou empresas, são deixados à mercê da capacidade dos seus ISPs detetarem e os notificarem de tais ataques. Sendo esse o caso, esta dissertação propõe uma plataforma capaz de monitorizar a conectividade entre redes de modo a detetar ataques de encaminhamento BGP. Esta plataforma foi construída de forma a ser o mais modular possível, de modo a facilitar a alteração ou adição de novos métodos de deteção de anomalias. A plataforma no estado atual tem já integrada duas metodologias. Uma das metodologias é baseada em um artigo já publicado, sendo a outra proposta pelo autor desta dissertação. Desde recolha de dados utilizando várias sondas, à sua análise de modo a detetar possíveis anomalias, tudo isto será apresentado e explicado de maneira a demonstrar que a plataforma proposta é realmente capaz de detetar em tempo útil este tipo de ataques, com uma precisão superior a 90%.

Esta plataforma pode então ajudar o utilizador a defender-se contra estes ataques, dando a informação de quando estes ataques estão a ocorrer, quase em tempo real, permitindo também que os utilizadores possam empregar contra medidas que serão acionadas automaticamente pela plataforma, caso estejam ativas, oferecendo assim um maior controlo aos utilizadores e menor dependência dos ISPs.

Abstract

In order for Internet connectivity to be possible, routing protocols have been created to assist in this task. The global routing protocol in use is BGP, which uses the aggregation of several network prefixes into ASes to create a graph containing information regarding routes to all public network prefixes, leading to global connectivity. Despite serving its purpose, this protocol is based on blind trust between all the BGP peers and as such leaves it exposed to attacks. Since this protocol is responsible for global connectivity, an attack carried on this protocol can have traffic re-routed from its normal path, and right into the attackers' hands, which may then be able to read and or alter the information contained in the traffic. Although security measures have been created for this protocol, they are not widely deployed, and, as such, most of the BGP devices' routing tables can still be compromised by a rogue BGP peer.
ISPs have ways to detect these kind of attacks, and act upon them but the users, such as private users or companies, are left at the mercy of their ISPs ability to detect and notify their clients of such attacks. That being the case, this dissertation proposes a platform capable of monitoring networks in order to detect BGP routing attacks. The platform has been made as modular as possible, to facilitate changes, and addition of new methods to detect such anomalies, and has also implemented two different methodologies for the detection of BGP routing anomalies. One of them based in an already published paper while the other one is proposed by the author of this dissertation. From data collection with the use of several probes, to the analysis of said data to detect the anomalies, all of that will be presented and explained to demonstrate that the platform does indeed detect BGP routing attacks with an accuracy of over 90%.
This platform can then help the users to defend themselves against such attacks, by providing information of when those are happening in near real-time as well as allow for the deployment of custom countermeasures, which can be set to activate when an alarm is raised, giving more control to the users and making them less reliant on their ISPs for information and action.

Contents

List of Figures

List of Tables

Acronyms

AES Advanced Encryption Standard. 29, 31, 34, 52, 53

API Application Programming Interface. 28, 29, 31, 34, 36, 37, 43–45, 52, 59–61

AS Autonomous System. 3, 8–16, 20, 21, 23, 86–88

ASN Autonomous System Number. 3, 8, 10–13, 15, 16, 22–24, 47, 68, 87

AuthCert Authorisation Certificate. 13

BGP Border Gateway Protocol. 1, 3, 4, 8–17, 19–21, 26, 27, 29, 31, 37, 55, 65, 68, 70, 75, 86, 88, 91

BGPSec Border Gateway Protocol Sec. 13

CA Certificate Authority. 52

CPM Central Probe Module. 28–34, 36, 37, 42, 43, 45, 52, 88

CSS Cascading Style Sheets. 54

DNS Domain Name Servers. 14

EBGP Exterior Border Gateway Protocol. 11, 12

GHz GigaHertz. 53

GRE Generic Routing Encapsulation. 65, 66

GUI Graphical User Interface. 29, 31–34

HTML Hypertext Markup Language. 54

HTTP Hypertext Transfer Protocol. 28, 29, 52, 62

HTTPS Hypertext Transfer Protocol Secure. 29, 52, 62

IANA Internet Assigned Numbers Authority. 8, 10

IAR Internet Alert Registry. 15

IBGP Interior Border Gateway Protocol. 11, 12

ICMP Internet Control Message Protocol. 23, 30, 46, 68

IP Internet Protocol. 8, 10, 11, 31–34, 37, 40, 41, 43, 45, 47, 50, 51, 55, 58, 62, 65–67, 70

IPv4 Internet Protocol version 4. 3, 8, 12, 14, 22, 65, 66

IPv6 Internet Protocol version 6. 3, 8, 22

ISP Internet Service Provider. 3, 4, 14, 15, 19, 20, 26, 65

JSON JavaScript Object Notation. 33, 37, 38, 40, 41, 44, 45, 47, 48

KVM Kernel-based Virtual Machine. 53, 63

LAN Local-Area Network. 1

MITM Man In The Middle. 15

MP-BGP Multiprotocol Border Gateway Protocol. 8, 12

MP_UNREACH_NLRI Multiprotocol Reachable Network Layer Reachability Information. 12

MP_UNREACH_NLRI Multiprotocol Unreachable Network Layer Reachability Information. 12

NL Network Layer. 12

NLRI Network Layer Reachability Information. 10, 12

OpenVZ Open Virtuozzo. 63

ORM Object-relational Mapping. 60, 91

PHAS Prefix Hijack Alert System. 15

PKI Public Key Infrastructure. 13

REST Representational State Transfer. 28, 29, 52

ROA Route Origination Authorisation. 13

RPKI Resource Public Key Infrastructure. 13

RSA Rivest–Shamir–Adleman. 29, 31, 32, 34, 37, 40, 43, 44, 52, 53, 62

RTT Round Trip Time. 16, 20–24, 46, 47, 49, 68, 70, 72, 73, 76–79, 87

S-BGP Secure Border Gateway Protocol. 13

SCP Secure Copy. 28, 33, 39, 42, 52

soBGP Secure-Origin Border Gateway Protocol. 13

SSH Secure Shell. 28, 29, 32, 37, 39, 40, 42, 52, 62, 67

SSL Secure Sockets Layer. 52, 62

TCP Transmission Control Protocol. 8, 9, 68

TSV Tab Separated Values. 33, 48, 57

TTL Time To Live. 22, 23, 86

VoIP Voice over Internet Protocol. 51

VPS Virtual Private Servers. 28, 53, 63, 66, 67, 87, 88

WAN Wide-Area Network. 1

WSGI Web Server Gateway Interface. 62, 92

ZB Zettabyte. 4

Chapter 1

Introduction

1.1 Motivation

The Internet has seen an immense growth in the past 17 years, going from a user's penetration rate of 6.5% of the world population in 2000 [1] to 48% in 2017 [2]. In developed countries that rate is even higher (81.0%), with European countries having a penetration rate of 79.6%. This world-wide connectivity is paramount in today's world, be it for social media, banking systems, researchers, hospitals, power grid, water supply and more. Almost real-time communication also allows us to be connected, no matter the distance, to whomever we decide, as long as they have access to the Internet. This means that there is an incredible amount of data going through the Internet at any given second and, although some of it may not be considered sensitive, some of it may be such as financial, business operational, critical infrastructures control, corporate business plans, etc., and some of that data may not always be secure.

To understand how connectivity is possible and the Internet is 'built', one needs to have some knowledge of the underlying technologies and protocols used to accomplish that result. The physical elements that a network is composed of are two or more terminals, such as a computer, that are somehow connected; the connection may be wired, or wireless, but as long as both terminals can communicate with each other, they are part of a network. Now, in the global scheme of things, a network is far more than several terminals connected directly to each other. There are several types of networks, namely Local-Area Network (LAN) and Wide-Area Network (WAN). LANs are normally deployed within a geographically close group of terminals, such as a home, or small business. WANs are what puts together what is referred to as the Internet; they are used to create massive networks of billions of terminals, and these are the type of networks that will be explored in this dissertation. For communications to be possible between different LANs or even WANs, they must know how to reach one another, and that is where the routing protocols come into play. There are several, but the one that this dissertation will explore is Border Gateway Protocol (BGP) as it is the only protocol for global routing. BGP allows for different networks to know the route to take to get to their intended destination; the protocol makes use of Autonomous System (AS) as a way to sort network prefixes into groups. Think of it has a neighbourhood, every house has a different number, but the postal code is the same. Several network prefixes can belong to the same AS,

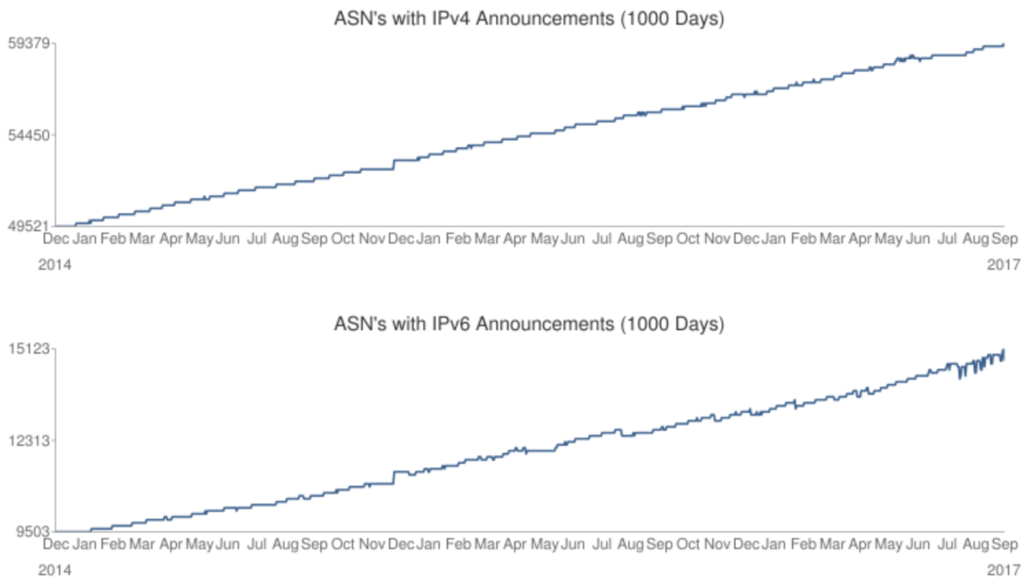


Figure 1.1: Autonomous Systems adverts.
Source: Hurricane Electric

and those are the prefixes that will be announced to the neighbouring BGP partners. There has been an increase in the number of adverts from different Autonomous System Number (ASN)s in the past years, as well as an increase in the Internet Protocol version 4 (IPv4) and Internet Protocol version 6 (IPv6) network prefixes contained within those adverts (Figure 1.1 and Figure 1.2). This information is growing each day, which means that queries on databases containing this information will become more intensive.

Pilosov & Kapela [3] proposed an exploit of the Border Gateway Protocol (BGP) vulnerabilities to implement BGP routing redirection attacks. In recent years, several reports [4, 5, 6] describe evidences of active traffic redirection attacks. Because BGP provides the path from point A to point B, if this information were to be compromised, one could redirect traffic to or from a network in order for it to take a detour from point A to point C and then B instead of the normal path from point A to B. What this means is that someone could potentially be redirecting, capturing and perhaps modifying some of the traffic and, sometimes, for nefarious reasons. This is not only theoretical since it has already happened in the past, such as when the United Kingdom's Atomic Weapons Establishment's traffic was being redirected through Ukraine in 2015 [6] which is depicted by Figure 1.3. These attacks represent a security risk, not only for companies but potentially for individuals as well.

This kind of attack is possible by gaining access to the credentials or terminal of an Internet Service Provider (ISP) network manager. By doing so, the attacker can then freely alter the BGP adverts to the remaining peering neighbours and redirect the traffic to a new destination. This traffic can then be captured, resent, altered or just dropped. Although most commonly the traffic is captured and resent without changes so that the users under attack do not notice anything is happening. Figure 1.4 shows such scenario. With the amount of data

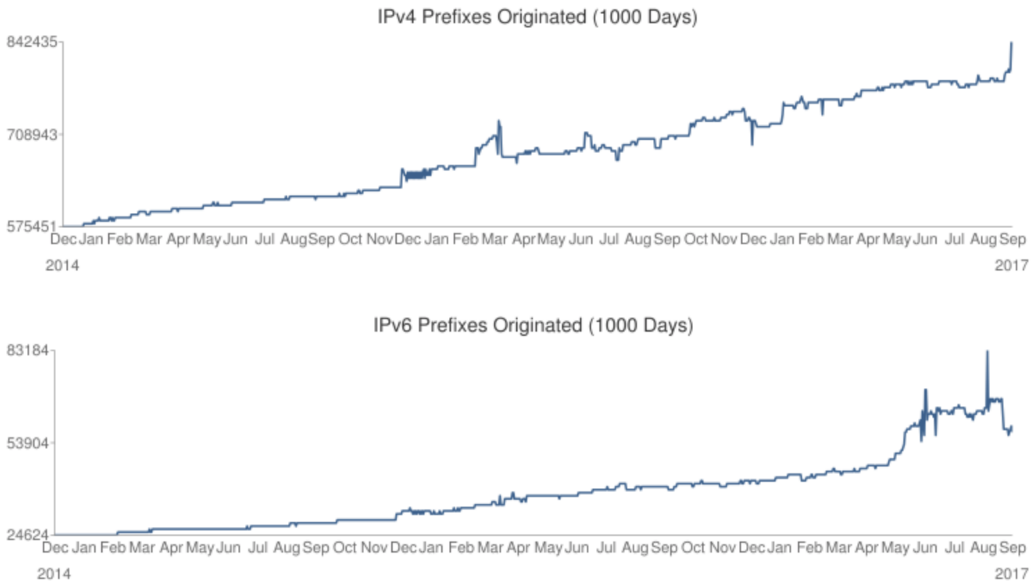


Figure 1.2: IP addresses prefixes.
Source: Hurricane Electric

that gets transmitted each year, which was 1.2 Zettabyte (ZB) in 2016 [7], the temptation to steal or modify all this information only increases; ISPs have ways to verify if a BGP redirect attack has happened or is happening, by verifying their routing tables, BGP adverts and purging them to correct the values if that's the case. Another solution would be for ISPs to deploy BGP neighbour authentication using MD5 [8], but for that to happen, all peers would have to agree to deploy the authentication. Since it is not a widespread practice at the time of the writing of this dissertation, the problem remains. The average company (and user) is then left at the mercy of the ISPs since without the ability to control BGP and no access to the ISPs' routing tables, little tools or information remain to assist them in the detection of such attacks. Information theft or modification is not only a privacy concern, but it may have financial consequences for the victims involved and, as such, a solution to at least be able to detect these attacks for the regular user is proposed.

1.2 Objectives

As mentioned above, this dissertation proposes to create and implement a method of detecting BGP routing attacks. The objective is to develop a modular platform able to detect BGP routing attacks without relying on ISPs' routing information, with meagre costs of deployment and operation.

1.3 Document Outline

This document is organised as follows:

- **Chapter 1** is the introduction of the work.

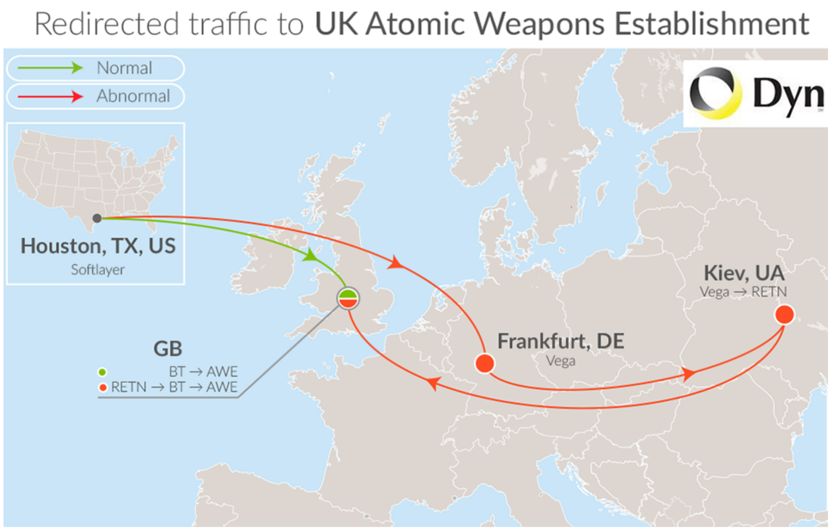


Figure 1.3: Redirected traffic to UK Atomic Weapons Establishment.
Source: www.dyn.com

- **Chapter 2** presents the context and state of the art.
- **Chapter 3** presents the problem and methodologies.
- **Chapter 4** presents the architecture and system design.
- **Chapter 5** explores how the system was implemented.
- **Chapter 6** contains the results obtained with the system.
- **Chapter 7** presents the conclusion and future work.

1.4 Scientific Contribution

A paper partially related to this dissertation has been accepted on the ICISSP 2018 Conference on Information Systems Security and Privacy. At the time of writing of this document, the proceedings were still not available online.

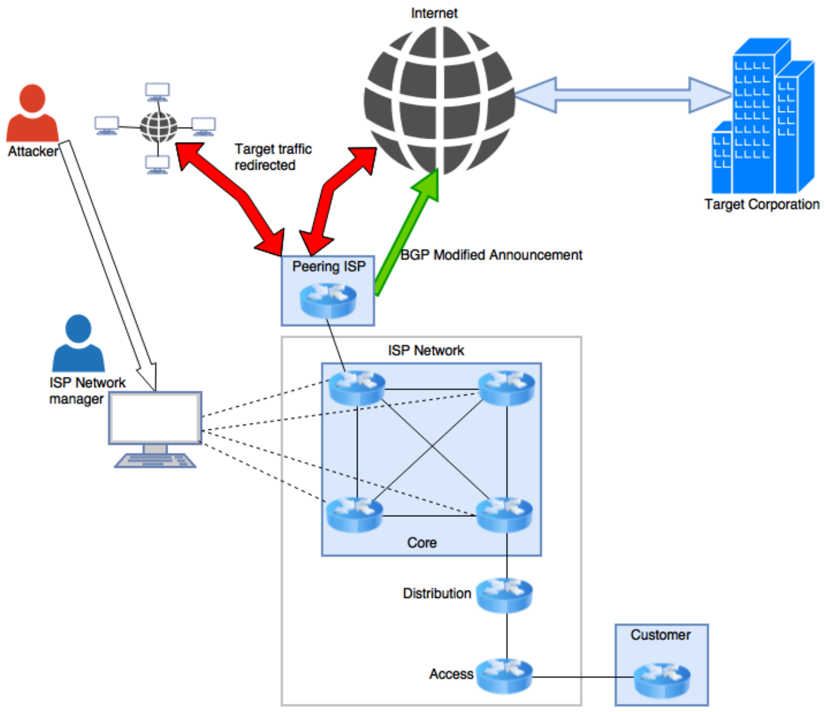


Figure 1.4: How a BGP attack occurs.

Chapter 2

Context and State of the Art

This chapter will present some fundamental concepts needed to understand the work presented in this dissertation, such as how Internet connectivity is achieved and the routing protocols used to achieve that connectivity as well as the current protocol constraints, attacks performed by exploiting the protocol vulnerabilities, the methods that can be utilised to detect and stop (when possible) such attacks.

2.1 Context

The Internet that is available to most of us is built on a number of terminals and network equipment, connected physically by different types of cables and or waves but, in order to establish true connectivity, these devices must know how to reach one another, no matter where they are physically located. To do so, the terminals and network devices must know where to send the traffic to and, for that to be possible, routing protocols were created. A routing protocol is an algorithm, or set of rules, that allows network equipment, in this case routers (a router is a network device that forwards packets between at least two different networks utilising the headers in the packets and its routing tables), to determine the most appropriated path to which send traffic towards a given destination. With this information, routers can obtain information regarding the several network prefixes with whom it is required to communicate. There are two types of routing protocols: internal routing and external routing. The former is used within the same network or network prefixes; it is mostly local (logical); the latter is used between different networks or network prefixes, between different logical locations. The routing protocol that is the focal point of this dissertation is Border Gateway Protocol (BGP) and its successor Multiprotocol Border Gateway Protocol (MP-BGP) which adds, amongst other things, support for Internet Protocol version 6 (IPv6) while the former only supports Internet Protocol version 4 (IPv4) [9].

"Border Gateway Protocol is an inter-autonomous system routing protocol designed for TCP/IP internets" [10]. To understand how BGP works, first one must understand what an **Autonomous System (AS)** is. "An Autonomous System (AS) is a connected group of one or more IP prefixes under the control of one or more network operators which has a SINGLE and CLEARLY DEFINED routing policy" [11]. An **AS** is then composed of one, or more, network prefixes which will then be advertised to the neighbouring BGP peers in

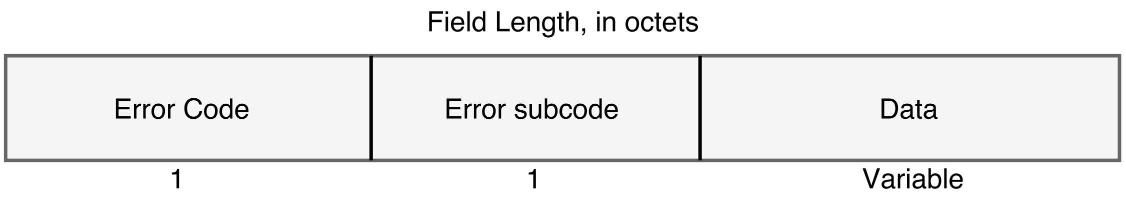


Figure 2.1: BGP NOTIFICATION message

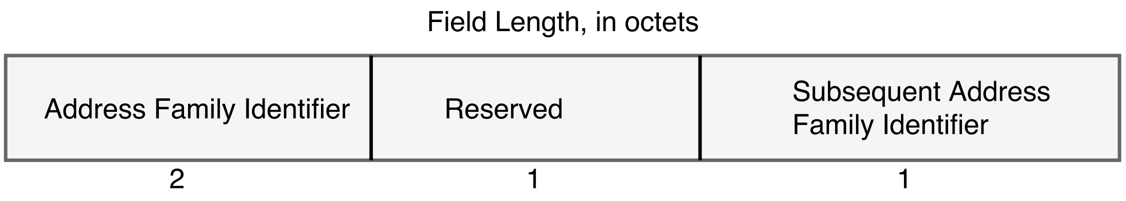

Field Length, in octets
Address Family Identifier
Reserved
Subsequent Address Family Identifier
2
1
1

Figure 2.2: BGP ROUTE-REFRESH message

order to establish connectivity with one another. Each AS that desires to have its routes advertised to the Internet must, without exception, obtain an ASN from Internet Assigned Numbers Authority (IANA). These ASNs were 16-bit (2 octets) long up until 2007 when the field size was increased to 32-bit (4 octets) [12]. The former has a range from 0 to 65535 while the latter has a range from 0 to 4294967295. In order to represent these values three representation formats were proposed: asplain, asdot+ and asdot [13]. Asplain represents the ASN in decimal integer notation while asdot+ uses a notation of two 16-bit integer values joined by '.' ($x.y$) where the ASN 65535 would be represented as 0.65535, 65536 as 1.1 and so on up until 65535.65335 which is the highest value possible; finally, the asdot defines the syntax as using asplain for ASNs up until 65535 and asdot+ for all above that. Now that it is known what an AS is and how they are identified (by their ASN), the next step is to understand the messages that are exchanged between the different ASes and their respective BGP peers.

BGP has 5 types of messages [14], [15], with a maximum size of 4096 octets, with the BGP header being 19 octets long, those types are:

- 1 - OPEN
- 2 - UPDATE
- 3 - NOTIFICATION
- 4 - KEEPALIVE
- 5 - ROUTE-REFRESH

Before any AS information can be traded between BGP enabled devices, they must first establish a Transmission Control Protocol (TCP) [16] connection between them. After a TCP connection has been established, an OPEN message is sent to the peers. This message is composed by the BGP message header (Figure 2.3, which contains the fields *Marker*, *Length*, *Type*, and the OPEN message, which contains the fields *Version*, *Autonomous System*, *Hold*

Time, *BGP Identifier*, *Opt Parm Len* and *Opt Params*). The first field (*Marker*) is a 16 octet field that is included for compatibility purposes and must be set to all ones; the second field (*Length*) is a 2 octet unsigned integer which indicates the total length of the message, including the header itself, in octets; the *Type* field is a 1 octet unsigned integer that indicates the type code of the message, '1' for OPEN, '2' for UPDATE, '3' for NOTIFICATION, '4' for KEEPALIVE and '5' for ROUTE-REFRESH; *Version* is a 1 octet unsigned integer that indicates the protocol version number of the message; *Autonomous System* is a 2 octet unsigned integer that indicates the ASN of the sender (when the sender is using a 4 octet ASN, the value will be "23456" by default, as stated by IANA, which is reserved and designated as AS_TRANS [17]); *Hold Time* is a 2 octet unsigned integer that indicates the number of seconds the sender proposes for the KEEPALIVE messages frequency, using the smallest *Hold Time* between them (this value must be either 0 or at least 3 seconds [14] although CISCO allows for a smaller window of 1 second [18]); *BGP Identifier* is a 4 octet unsigned integer that identifies the sender (this value is set by the sender itself, commonly to an IP address belonging to the sender, which is determined at start up and is the same for every local interface and all BGP peers); *Opt Parm Len* is a 1 octet unsigned integer that indicates the total length of the *Opt Params* field in octets (if set to 0, there are no optional parameters); *Opt Params* contains a list of optional parameters, each of them encoded as Parameter Type, Parameter Length, Parameter Value (the first two have a length of 1 octet each, while the last has variable length); the minimum size of the OPEN message is 29 octets long (including the message header). After this message is sent and accepted by the peers, KEEPALIVE messages are sent periodically to all the peers, unless the *Hold Time* is 0, in which case no KEEPALIVE messages are exchanged.
Now that the peering is complete, it is possible to exchange routing information between peers. That exchange is done via the UPDATE message type (Figure 2.4), whose information can then be used to construct a graph describing the relationships of different ASes. Not unlike the OPEN message, this message type also contains the message header previously described, as well as its own fields: *Unfeasible Routes Length*, *Withdrawn Routes*, *Total Path Attribute Length*, *Path Attributes* and *Network Layer Reachability Information (NLRI)*. *Unfeasible Routes Length* indicates the length of the withdrawn routes field (a value of 0 means that there are no routes being withdrawn and as such the *Withdrawn Routes* field is not present). *Withdrawn Routes* contains a list of IP address prefixes for the routes being withdrawn. *Total Path Attribute Length* indicates the length of *Path Attributes* field (a value of 0 means that there are no new routes and that the field *Path Attributes* is not present). *Path Attributes* contains the characteristics of the advertised path. This former field has several possible attributes:

- ORIGIN
- AS_PATH
- NEXT_HOP
- MULT_EXIT_DISC
- LOCAL_PREF
- ATOMIC_AGGREGATE

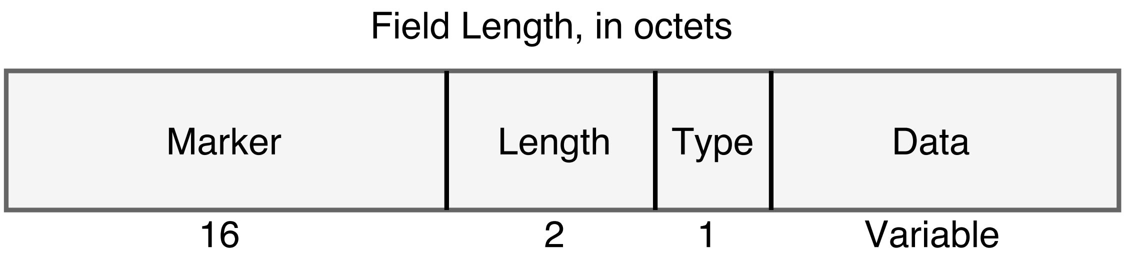


Figure 2.3: BGP message header

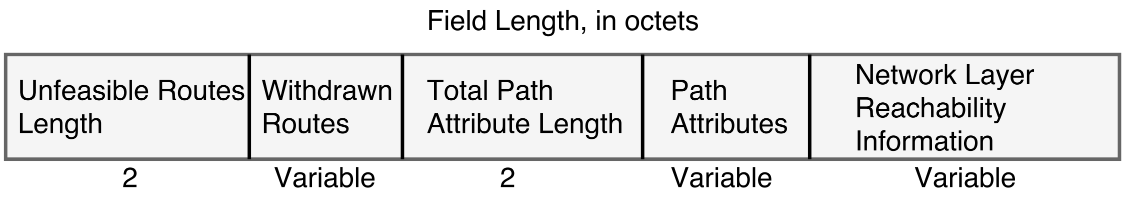


Figure 2.4: BGP UPDATE message

- AGGREGATOR

The *ORIGIN* attribute is mandatory and defines the origin of the path's information, so the other BGP peers know the origin. *AS_PATH* is a mandatory attribute composed of a sequence of ASN path segments. *NEXT_HOP* is also mandatory and defines the IP address of the border router to be used as the next hop to the destination(s) listed in the *NLRI* field. *MULT_EXIT_DISC* is optional, and it is used to discriminate between multiple exit points to a neighbouring AS. *LOCAL_PREF* is not mandatory but should be included in all UPDATE messages that are sent to internal peers, and it is used to select the preferred routes (the higher the value, the higher the preference) but it must not be sent to external peers, except in the case of BGP Confederations [19] (this attribute should be ignored by the receiver if the exceptional case mentioned before is not met). *ATOMIC_AGGREGATE* is a discretionary attribute that should be included when the aggregate excludes at least some of the ASNs present in the path (this attribute should not be removed when propagating the route to other peers). *AGGREGATOR* is an optional attribute that contains information about aggregate routes. Finally, the last field of the UPDATE message is *Network Layer Reachability Information* which contains the list of IP address prefixes for the advertised routes. The last two fields of the UPDATE message are the ones that are most important in the context of this dissertation since it is within them that the routes for the different networks are transmitted to the peers.
The next message type is the NOTIFICATION message, which is sent when an error condition is detected. Whenever this message is sent, the BGP connection between the peers sending and receiving this message is closed immediately. The next message type, KEEPALIVE, contains only the header of the BGP messages and is used to keep the connection open. Finally, the last message type is ROUTE-REFRESH, which is used between BGP peers that support this type of message in order to request the routing information from its BGP neighbour whenever the BGP policies were changed (on the sender).
As previously mentioned, the UPDATE message type contains the routing information. When advertising a new route (or changes to the route) to other BGP peers, each AS on the path

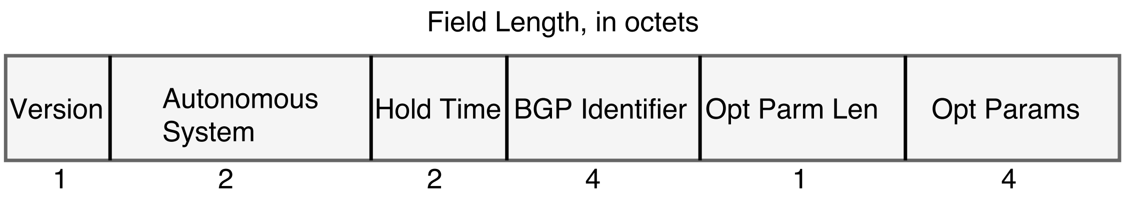


Figure 2.5: BGP OPEN message

will append its own ASN number on the *AS_PATH* field (Figure 2.6). By doing so, each of the peers can select a preferred route to a given destination. This selection process has 3 phases: (*i*) calculation of the degree of preference, (*ii*) route selection and (*iii*) route resolvability condition [14]. Phase *i* calculates the degree of preference for each route received from a peer; phase *ii* starts when phase *i* is complete, and it is responsible for choosing the best route for each destination; phase *iii* is invoked after phase *ii* and it is responsible for disseminating routes to each peer (route aggregation and information reduction can also be performed in this phase). The calculation of the preferred route depends on a number of factors, some of them only applicable when speaking of internal peers, but, since the internal routing is not the focus of the dissertation, the most important aspect of the route selection is the one where routes are learned from external peers. When presented more than one possible route to the same destination, without considering routing policy preferences, the order in which routes are chosen is [14], [20]:

- 1 - Prefer the path with higher LOCAL_PREF
- 2 - Shortest number of ASNs in the *AS_PATH* attribute.
- 3 - Lowest *ORIGIN* value.
- 4 - More-preferred *MULTI_EXIT_DISC*.
- 5 - Prefer Exterior Border Gateway Protocol (EBGP) over Interior Border Gateway Protocol (IBGP).
- 6 - More-preferred interior cost (metric to *NEXT_HOP*).
- 7 - Prefer the route announced by the BGP peer with the lowest *BGP Identifier* value.
- 8 - Prefer the route announced by the lowest BGP peer address.

The order presented on the list above is the actual order by which routes are selected. The first item, *LOCAL_PREF*, is used in IBGP and the highest the value, the more preferred that route is. The next two items (2,3) are self explanatory, it prefers a route with a shorter number of ASNs and if there is a tie, the one with the lowest *ORIGIN* value is selected. Item 4 only applies between routes learned from the same neighbouring AS (routes that do not have the *MULTI_EXIT_DISC* are considered to have the lowest possible value). As for item 5, from all the remaining possible routes the ones learned via IBGP are discarded in favour of EBGP ones. If the tie still remains (item 6), the route with a lower cost to a reachable *NEXT_HOP* is preferred (if the *NEXT_HOP* is reachable but the cost can not be determined, then this step is skipped). If the tie is still not resolved, the next step (7) tries to break the tie

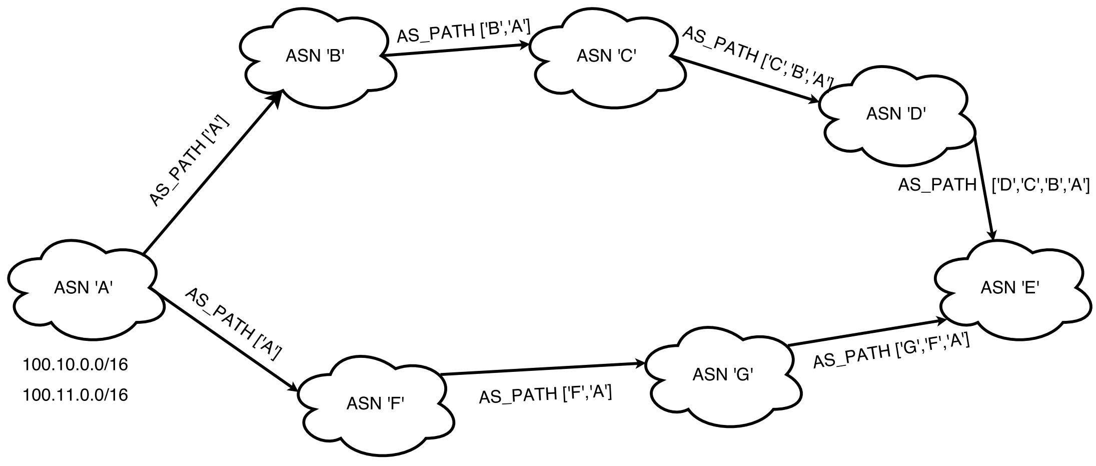


Figure 2.6: BGP Advertising example

by using the lowest *BGP Identifier* and step (8) uses the lowest BGP peer address. In order to aid in understanding the route selection process, let us take a look at a simple example with two routes being announced by the same AS to multiple ASes (Figure 2.6). AS 'A' advertises the route to its two networks (100.10.0.0/16 and 100.11.0.0/16) to AS 'B' and AS 'F', appending its own ASN to the path. Upon receiving this message, both ASes ('B' and 'F') now know that the route to the two network prefixes can be through ASes 'A'. When forwarding this information to their respective neighbours, both will append their own ASN number to the path, until it reaches AS 'E'. AS 'E' now has to make a routing decision: send traffic towards 100.10.0.0/16 and 100.11.0.0/16 via AS 'D' or AS 'G'. In this case, and disregarding any policy routing preferences, the choice is clear: choose the path with the lower number of ASes which is, in this case, via AS 'G'. This is a simple example of routing selection when using BGP, but it plays an important role in the possible vectors of attack that can be performed exploiting this routing protocol.

All the information previously presented is in regard to BGP, but most of it applies to MP-BGP as well. The only three pieces of information that are IPv4 specific in the BGP protocol are: (*i*) *NEXT_HOP*, (*ii*) *AGGREGATOR* and (*iii*) *NLRI*. In order to be able to use multiple Network Layer (NL) protocols, two changes are necessary: (*i*) ability to associate a particular NL protocol with *NEXT_HOP* information and (*ii*) ability to associate a particular NL with *NLRI* [9]. For these two changes to be implemented, and backwards compatibility ensured, two new attributes were introduced into the UPDATE message: *Multiprotocol Reachable Network Layer Reachability Information (MP_UNREACH_NLRI)* and *Multiprotocol Unreachable Network Layer Reachability Information (MP_UNREACH_NLRI)*. The former carries the set of reachable destinations, as well as the information regarding the *NEXT_HOP*, and the latter carries the set of unreachable destinations (the *NEXT_HOP* attribute is not required here since routes are being withdrawn and not added). Both attributes are non-transitive, meaning that when a BGP speaker does not support multiprotocol capabilities, it will just ignore the extra information and will not pass it on to other peers [9]. Both these attributes, when present, are in the *Path Attributes* of the UPDATE message (Figure 2.4). Since that field is of variable length in BGP, this ensures that the protocol is not broken when utilising MP-BGP. If a router supports MP-BGP, that information will be provided when establishing a connection to another peer, via the OPEN message type (Figure 2.5) on the *Opt Params* field. These changes then allow for the use of multiprotocol route advertisement, while keeping the protocol intact and without the need for a whole new routing protocol.

2.2 Protocol constraints

This protocol relies on blind trust. As seen above, all that is required to advertise routes to the Internet is an established connection with other BGP peers. Although there are some preventive measurements in place, such as routing loops detection (an AS BGP router will not accept any routes that contain its own ASN), there are no currently adopted mechanisms that authenticate a sender or validate the truthfulness of its information. With that being the case, any router that has established BGP relations with other peers can advertise any route, with the respective peers accepting that information as truthful and propagating those changes to their respective neighbours, and so on. This leaves the protocol open to misinformation, with no way to automatically detect it. This can be then exploited to serve the purposes of the attackers.
Some attempts to solve this issue have been presented, such as: Secure Border Gateway Protocol (S-BGP) [21], Secure-Origin Border Gateway Protocol (soBGP) [22] and Border Gateway Protocol Sec (BGPSec) [23]. S-BGP consists of four major elements:

- A Public Key Infrastructure (PKI) [24] representing the ownership and delegation of address prefixes and ASN.
- *Address Attestations* which the owner of a prefix uses to authorise an AS to originate routes to said prefix.
- *Route Attestations* created by an AS allowing a neighbour to advertise prefixes.
- *IPSec* for point-to-point security of BGP related traffic between peers.

With the changes introduced by S-BGP, peers would be able to validate the authenticity of the routing information received using the PKI for validation of the authorisation of other peers to represent ASes, the legitimacy to originate a route to prefixes (*Address Attestations*).
soBGP focus on two distinct goals, which converge to a broader goal: the legitimacy of the routing information. The first goal is to make sure that the AS originating the destination prefix is authorised to do so. The second goal is to verify if an AS advertising a destination prefix actually has a path to the AS originating that destination prefix. The first goal is accomplished by using an Authorisation Certificate (AuthCert), which would allow other peers to verify the signature of the sender, ultimately verifying its identity and if it is authorised to advertise those address prefixes. The second goal is accomplished by building a topology map of the paths of the entire internetwork, built by each connected AS.

BGPSec [23] was proposed to ensure the the correct transmission of the routing information between several routers, which has no cryptography assurance in soBGP [23], since the manipulation of the *AS_PATH* is still possible. To solve this, BGPSec would give routers the ability to a sign the UPDATE messages and authenticate them with the usage of a Resource Public Key Infrastructure (RPKI) [25], a Route Origination Authorisation (ROA) [26] and an additional element of certification, *"router certificates"* [23], which signature can be interpreted as a signature from an authenticated router of an AS.

The problem with all of these solutions is one and the same: they are not widely adopted. This continues to leave BGP vulnerable to attacks.

2.3 BGP routing attacks

An attack on the BGP routing protocol can cause traffic to be deviated from its 'normal' path and into the hands of the attacker. Since there are no security measurements in order to prevent such to happen, it is fairly easy to execute this type of attack. As explained in section 1.1, to deploy an attack on the BGP routing protocol, the attacker has to gain access to the credentials or a terminal of an ISP network manager. Once that is accomplished, the attack can take place. To successfully hijack traffic, the attacker must use some of the properties of BGP to compromise it. Those properties are the ones regarding the selection of the path to take to a given destination. As previously explained, the route selection process prefers a shorter route by default, unless other policies are in place. It also prefers more specific network prefixes (bigger network mask) to less specific ones. With these two things in mind, it is possible to understand that the attack can be accomplished by doing one of these two things: advertise a route to the target destination with a shorter path and or advertise a more specific network prefix than the one currently being advertised by other peers. Let us take a look at an actual occurrence of the second case in the wild. In 2008, YouTube traffic started being hijacked by a Pakistani telecommunications company, Pakistan Telecom. YouTube owns the IPv4 network 208.65.152.0/22 (**AS 36561**), with the more specific 208.65.153.0/24 network hosting their Domain Name Servers (DNS) and Web servers (back in 2008), but YouTube was only advertising the more broad network prefix (208.65.152.0/22). Due to a government order, the Pakistani company started advertising a route for the more specific network prefix (208.65.153.0/24) to its own provider (AS 3491). This was then propagated to other ASes leaving YouTube service unavailable to multiple parts of the world [27]. This was most likely a bad configuration scenario and not really an attack, but it shows that it is possible to be done with relative ease. Some other attacks have occurred since such as the one previously referenced in section 1.1. This case resulted in the inability of the end users to access the service, leading them to notice that something was amiss. The attackers may not want to disrupt the traffic in order for the attack to continue going as unnoticed as possible. Alex Pilosov and Tony Kapela [3] presented a Man In The Middle (MITM) attack in 2008. The basic ideas were to hijack traffic but ultimately send it to the intended user and abuse the AS path loop detection feature of BGP to blind some of the ASes to the attack. This would allow an attacker to log, alter, misdirect or observe the victim's Internet traffic. Figure 2.7 depicts a route advert by the ASN 200 for its network prefix 10.10.220.0/22. The attacker (ASN 100) needs to decide a path through which the traffic will arrive at the victim after it has been redirected to himself. In this scenario, the attacker wants to affect the highest number possible of ASes, deciding to keep the path through ASNs 10, 20, 200 intact, while sending UPDATE messages to the remaining ASes containing the more specific network prefixes 10.10.220.0/23 and 10.10.22.0/23 via 10, 20, 200. The BGP peers will then decide that the best route is now through ASN 100 (since the network prefixes are more specific) and send the traffic destined to the prefixes advertised by ASN 200 through ASN 100 (Figure 2.8).

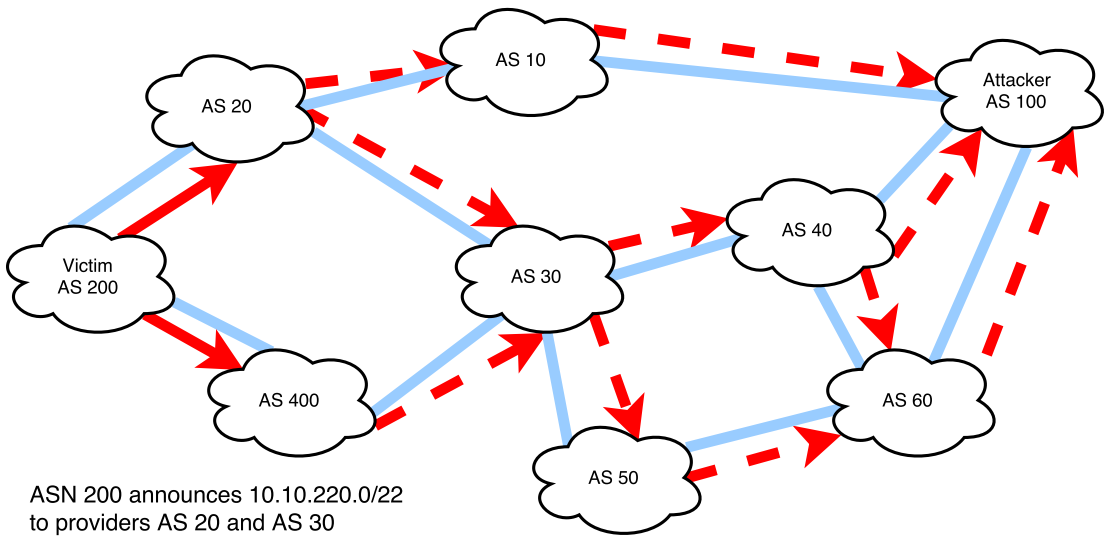


Figure 2.7: Route advert
Source: Pilosov & Kapela

The attacker has successfully hijacked the traffic towards ASN 200 from the ASNs 30, 40, 50 and 60. As can be seen, once an attacker has access to credentials or a terminal of an ISP network manager, the attack is simple to perform. Unfortunately, the current state of BGP usage has no way to stop these scenarios completely.

2.4 BGP routing attack detection

Since the solutions to mitigate BGP routing attacks previously presented are not widely adopted, the need for detection of such attacks is clear. Some monitors and or alarms exist for this purpose, such as Prefix Hijack Alert System (PHAS) [28], Internet Alert Registry (IAR) (discontinued), RIPE NCC MyASN (discontinued), BGPmon [29] and Renesys Routing Intelligence [30]. PHAS relies on BGP routing data collected by BGP collectors to detect announcements of prefixes belonging to the customer that are being advertised by a different origin AS.

"BGPmon uses a publish/subscribe overlay network to provide real-time access to vast numbers of peers and clients." [31]. It obtains information from BGP UPDATE messages and periodic routing tables snapshots which can then be used by clients. Clients, such as Cyclops [32], utilise the data provided by the publisher/subscriber service of BGPmon to detect routing anomalies, and activate the alarm when such are found [31].

NetViews is also a client of BGPmon, it uses the information received from BGPmon to infer a topology graph of the routing to different ASes [33].
Renesys Routing Intelligence, from what can be interpreted from the information provided on their website, also relies on BGP UPDATE messages to gather information and utilises said information to alert clients about possible routing anomalies. All the aforementioned rely, directly or indirectly, on data provided by BGP speakers (a router that announces BGP

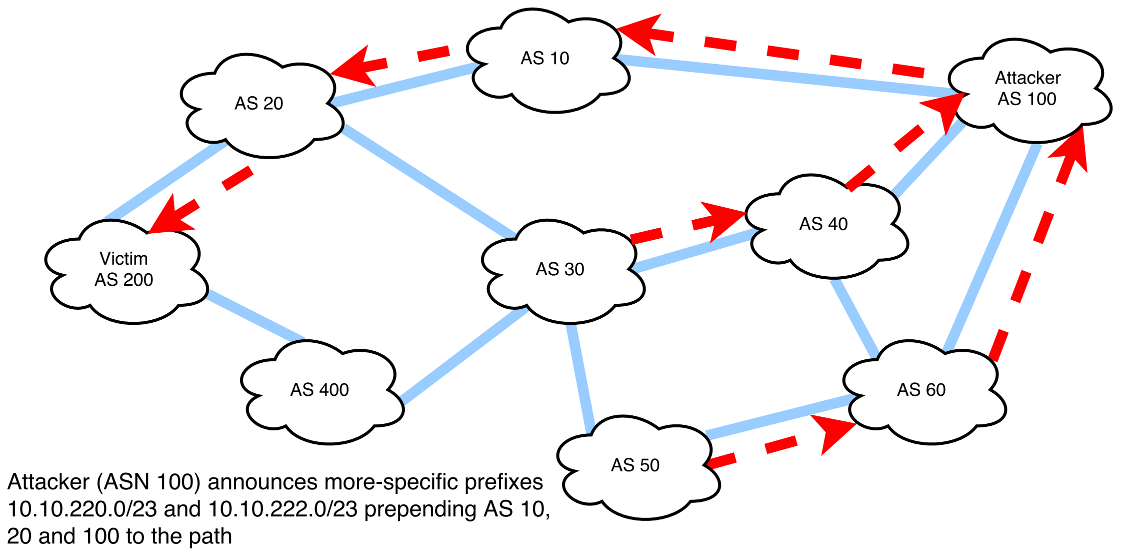

AS 10
AS 20
Attacker
AS 100
AS 40
Victim
AS 200
AS 30
AS 400
AS 60
AS 50
Attacker (ASN 100) announces more-specific prefixes 10.10.220.0/23 and 10.10.222.0/23 prepending AS 10, 20 and 100 to the path

Figure 2.8: Traffic hijack
Source: Pilosov & Kapela

information is considered a speaker).
There have been several proposals regarding the detection of BGP routing attacks, such as Classifying Anomalous Events in BGP Datasets [34], HEAP [35] and DARSHANA [36]. The first proposal utilises datasets of known malicious attacks as well as power outage events, and employ machine learning algorithms in order to detect routing anomalies. It makes use of Naive Bayes and Decision Tree models for the detection of anomalies. It is, however, reliant on information gathered by collectors of BGP UPDATE messages.

The second proposal aims to combine techniques from prior work [37], [38] to identify legitimate routing changes and remove those from the alarms in order to decrease the number of false alarms raised by the other systems. From what was possible to gather regarding [37] and [38], both methods rely on information that is obtained, directly or indirectly, from BGP UPDATE messages. This proposal focus is on utilising input from other detection systems, and attempt to reduce their rates of false alarms. Both of these proposals rely, in one way or another, on BGP UPDATE messages information, which is contrary to the goal of this dissertation.

The last proposal does not rely on that type of information, relying instead on data-plane information. It proposes several mechanisms used to detect routing anomalies: *Monitoring network latency (Lat)*, *Estimating hop count (Hop)*, *Calculating path similarity (Path)*, *Monitoring propagation delay (Prop)*. The first mechanism calculates the Round Trip Time (RTT) from a given source to a given destination. The second mechanism counts the number of intermediate devices between a given source and a given destination (hops). The third obtains the ASN for each of the hops and then compares the similarities of the path between different measurements. Lastly, the fourth mechanism attempts to isolate the propagation delay from the RTT value by calculating the average geographical distance between each of the hops and subtracting the estimated delay time from the RTT value. This proposal is similar to this dissertation's goal, in the sense that it does not rely on BGP information. However, the second mechanism may induce errors, since it is not abnormal for legitimate routes to have several different paths in the same ASN, either because of routing policies or load balancing mechanisms. Also, this dissertation proposes a modular platform cable of detecting BGP routing anomalies, and, as far as it is possible to determine from the references used, this proposal does not provide such platform, although it does propose possible detection mechanisms.

An implementation aimed towards routing anomaly detection without the use of BGP UPDATE messages already exists, it was developed as part of a dissertation [39]. This implementation makes use of terminals to monitor target network prefixes, and implements the methodology presented in [40]. This solution, however, does not allow for easy integration of new methodologies for both data capture, and data analysis.

An article published in 2005 [41] presents an algebraic theory for the study of the convergence of dynamic routing protocols, such as BGP. It presents possible algebraic theories that would allow for protocol convergence on all networks as well as convergence to optimal paths. This theory could be helpful in the creation of methodologies to detect routing anomalies in BGP.

Chapter 3

Routing anomaly detection

This chapter consists of two sections, which explain the main objectives and challenges of this work as well as how the data is obtained and analysed.

3.1 Problem overview

BGP routing attacks can go undetected by the end user due to lack of tools, and access to the ISP routing tables. This can cause traffic to deviate from its original path, possibly sniffed, compromised, and stolen. The ramifications of such attack may be harmless, irrelevant or inconsequential in some cases but when sensitive data is in question, the effects may be nefarious. The attack may cause all traffic to be dropped, never reaching its intended destination. This can cause critical operations to be halted, possibly resulting in financial loss. A similar case has occurred, although the routing anomaly was most likely not due to criminal intent when the traffic for YouTube was wrongly redirected through an ISP in Pakistan [27], which resulted in all traffic to YouTube from certain ASes to never reach the destination. Other attacks do not drop the traffic that is being hijacked, but instead redirects it so it can be captured, and possibly sniffed or altered. That was the case when outbound traffic from a nuclear weapon research centre in the United States of America, bound to the United Kingdom, was redirected through Kiev (Ukraine). This type of attack poses a possible security risk since these are sensitive communications about nuclear weaponry research. These attacks can completely disrupt communications on a global scale but, perhaps worse, they can go on undetected for extended periods of time, possibly leaving sensitive information up for the taking by malicious parties.

These attacks can happen anywhere along the path from source to destination. This does not mean, however, that the traffic is redirected towards the attacked router(s) since the attacker would be able to redirect the traffic from that router to another terminal as long as the attacker can successfully propagate the new routing path to the router's BGP peers. There are four case scenarios of BGP routing attacks that one needs to contemplate to fully understand the possibilities of an attacker (Figure 3.1). The first one (D) is that if the attack is redirecting traffic to a path close to the destination, the difference in the data obtained by the monitoring probes may be minimal, and the attack may pass undetected. The second case is similar to the first, but the difference is that the attack is occurring close to one, or

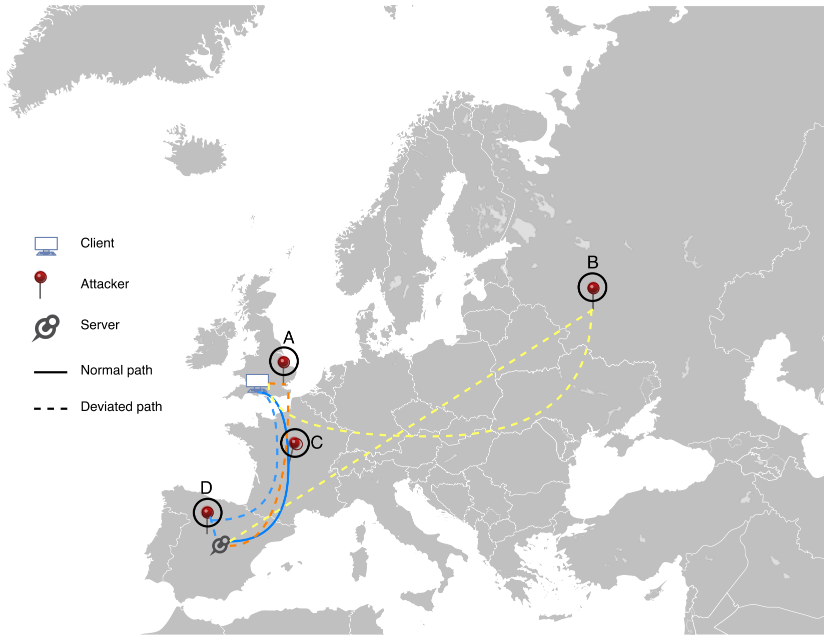


Figure 3.1: BGP routing attacks types.

more, probes (A). The third case is when the attack is happening within the normal path of the traffic (C) and lastly, the fourth and easiest to detect, is when the attack happens far away from the normal path, in which case it is easier to detect, due to the effect it has on the data that can be collected while monitoring the traffic (B). It is also worth mentioning that not all routing anomalies are an attack. It is not unusual for routing changes to happen, either due to load balancing or new policies between different ASes. These are what can be called false alarms in the context of the detection of routing anomalies. It will be explained further on, in the following chapters, how the platform attempts to attenuate such events. In the subsections below, the two methodologies proposed in this dissertation used to detect such attacks will be explained.

3.2 Methodologies

Traffic that flows on the internet does so at a variable speed, passing by multiple network equipment until reaching its destination. The speed at which the traffic flows is dependent on the physical layers, such as routers and cabling. It is also affected by the physical limits of signal propagation. This means that every packet must go through a path, and it will take T seconds to arrive at the destination. The time a packet takes to go from its source to the destination and back is designated as RTT. We also know that a packet will pass through multiple network equipment, from now on referenced as *hops*, along the way to and from its destination. To get this information, several probes, spread around the globe, were used to obtain measurements (Figure 3.2). The reason for several probes spread around the world is that if they are too close to their target, they might also be affected, and the attack may go unnoticed, but if there are several probes in geographically different places, the chance of the attack going unnoticed by all of them is decreased. With this information, two different, but

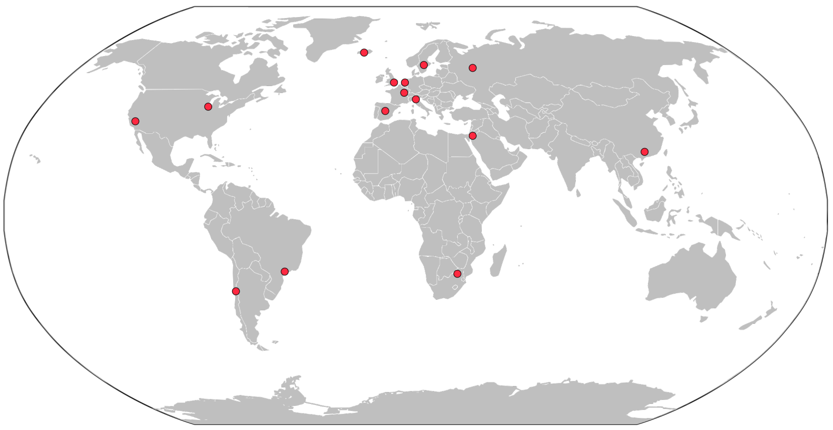


Figure 3.2: Location of all the CPMs spread around the globe.

related, ways of detecting a BGP routing anomaly, without relying on ISPs' routing tables or BGP adverts are proposed in this dissertation. There are two proposed methodologies which can be seen in the subsections below. The first methodology utilises information regarding RTT values to detect possible routing anomalies. The second methodology makes use of the path of ASes through which traffic flows as well as the RTT values obtained from the source to each of the hops in the path. The next section will explain both of these methodologies in detail.

3.2.1 Methodology 1

The first proposed method is to collect and analyse data regarding the RTT from probes to targets, as proposed in [40]. It relies on a number of probes P and monitored networks N (which will be addressed as simply *targets* from now on). The probes will monitor the targets every T seconds and gather the RTT measurements obtained for each of them. This data will then be used to detect possible anomalies. The time sequence is given by $\{T_{p,n,i}\} p \in P, n \in N$ where $\{T_{p,n,i}\}$ represents the i^{th} RTT measurement taken by a probe p to a target n. This sequence will be used to compare the new measurements and verify if there is an anomaly currently occurring. A local anomaly, on instant i, is then given by

$$A_{p,n,i} = \prod_{j=i-K+1}^{i} T_{p,n,j} > \epsilon \cdot \theta, \theta = \frac{1}{H} \sum_{j=i-K-H+1}^{i-K+1} T_{p,n,j}$$

, where H represents the size of the past measurements used to calculate the average RTT value, and K represents the number of consecutive measurements that are being analysed. A local anomaly is then declared when K consecutive measurements are above $\epsilon \cdot \theta$, with $\epsilon > 1$, being θ the average value of the last J measurements. A global anomaly, on instant i, is given by

$$G_{n,i} = \left(\sum_{p} A_{p,n,i} \right) > \rho P$$

. This means that a global anomaly will be declared when the percentage of the probes that are in a local anomaly state are greater than ρP. The local anomaly status will revert to normal once K consecutive values are not flagged as anomalous, while the global anomaly status will revert to normal once the total number of probes detecting a local anomaly falls below ρP. From Figure 3.1, it is possible to see that some of the attacks can deviate traffic through a significantly longer physical path, for example when traffic deviates via Russia, which will result in higher RTT values. This methodology aims to detect such deviations by using a set of past values, compare new values with the average of the former, and verify if these exceed the set threshold (ϵ). This methodology then makes use of something that can not be directly controlled by an attacker: the limitations of the physical propagation speed of the data.

3.2.2 Methodology 2

The second methodology also relies on the use of probes to monitor networks, but the focus now shifts towards the path, rather than basing the decisions solely on the RTT. When a packet leaves the network, it will go through several hops until it reaches its destination (or fails to reach it), which can belong to different ASNs, depending on the path taken. In order to better understand how this would work, one should know how this data can be obtained and what can be extracted from it. To find the path that a packet takes, one can use provided tools such as *traceroute*, which will make use of an attribute contained in packets: Time To Live (TTL) for IPv4 and Hop Limit for IPv6 (for simplicity, both will be referred to as TTL). Whenever a router receives a packet, it will decrease the TTL by at least 1, forwarding the packet if the TTL is greater than 0 and discarding it otherwise. When a packet is discarded by a router, the router will send an Internet Control Message Protocol (ICMP) message back to the original sender, informing it that the packet was discarded before reaching its destination. Since the goal is to know the most information possible about the path that is being taken, it is required to know all the hops from the source to the destination and traceroute allows that by sending packets that start out with a TTL of 1 and increasing it until it reaches the destination. Figure 3.3 depicts how TTL can be used to find out the hops in between source and destination. Based on the principles presented, one could assume then that an attack could simply be detected by observing changes in the path taken. The problem is that routing changes within the same AS are not uncommon, either because of load balancing or simply new equipment or faulty equipment that was taken offline. Attackers often also attempt to hide their tracks, and they use TTL to do so. By increasing the TTL by N, the packet will keep being forwarded N more hops without being rejected and a message sent back, and the attack can then be hidden.

In Figure 3.4 it is possible to visualise how the attacker would hide its tracks from the users: when receiving the original packet, simply increment the TTL to account for the number of hops between the next legitimate router. By doing this, if the users being affected were to trace the path its packets were taking, they would not be able to see the attack happening. This hiding technique is not flawless, however. One can hide the true path through which the packets flow, but what the attackers cannot change is the time it takes for them to travel. A longer route, under the same bandwidth conditions, will always present itself with a higher RTT value than the shorter route. That being the case, it is possible to detect changes in RTT between hops that otherwise would appear normal. Let us take a look once again at Figure 3.4, in it are present the routers R1 through R3, which represent the legitimate path,

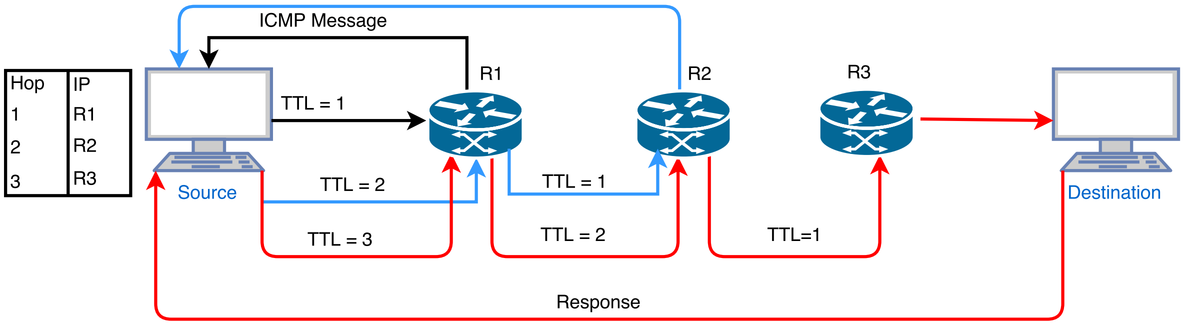


Figure 3.3: Sending packets with increased TTL.

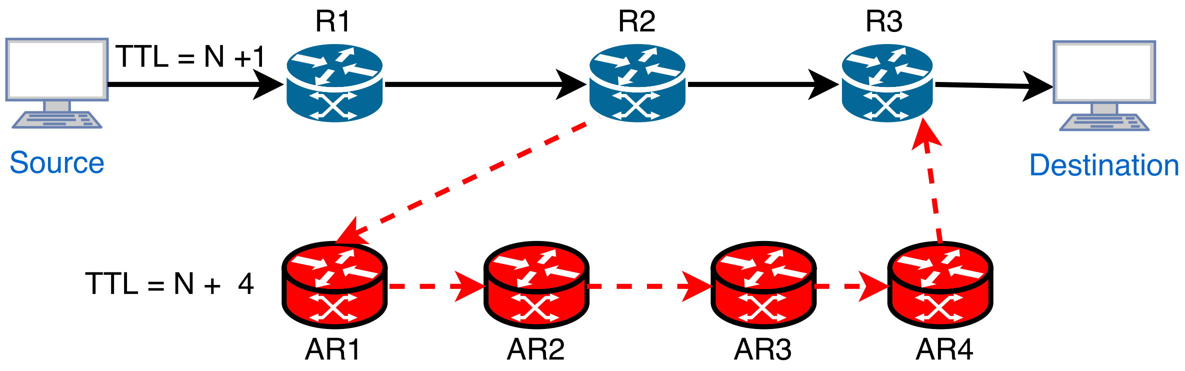


Figure 3.4: Hiding an attack by changing TTL.

and routers AR1 through AR4 which represent the path when under attack. If the attacker does not hide his tracks, then, with a simple traceroute, the user would be able to see that the path is now R1, R2, AR1, AR2, AR3, AR4 and R3 instead of the normal R1, R2, R3 path, but, if the attacker covers his tracks, the user would simply see the path as R1, R2 and R3. However, an increase in RTT could be noticed if the user had previous measures while not under attack. Now that we know what kind of information that can be extracted, there are several ways in which this information can be analysed: analyse each and every hop on the path, see if it differs from the previously known paths, and the RTT between each of them; analyse the changes that happen whenever the hops change ASNs and the respective RTTs between those changes. The proposed way to analyse it would be the second one, with the use of the ASNs jumps and respective RTTs. The reason for that being that routing changes within the same ASN are relatively common, and as such too many false positives could occur, resulting in an unreliable way of detecting the anomalies. However, since changes regarding the ASNs in the path taken by the traffic are far less common, the results would be more reliable than the one previously mentioned.

In order to implement the second methodology, let us assume that there are P probes that are part of a set P and monitored networks N that are part of set N. Let the sequence

$$\{H_{p,n,i}\}p \in P, n \in N, i \geq 0$$

contain the different ASNs in the path (in order from source to destination) for the i^{th}

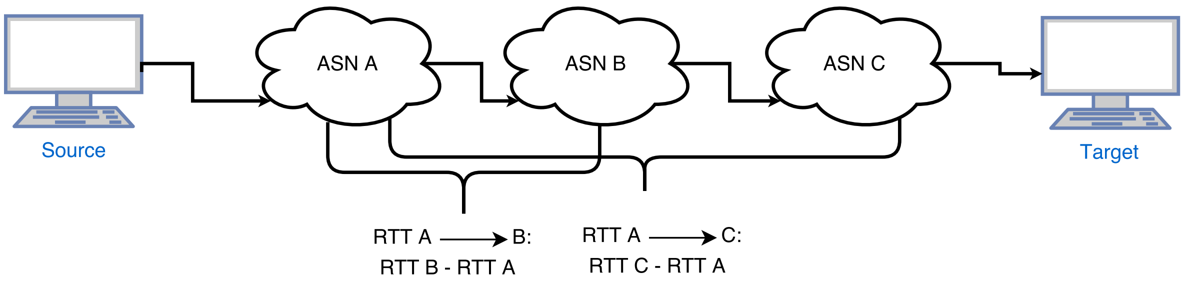


Figure 3.5: RTT between different ASNs.

measurement, and

$$\{R_{p,n,i,j}\}p \in P, n \in N, i \geq 0$$

contain the RTT values between the different ASNs hops in the path (Figure 3.5 depicts how the RTT for the sequence R is obtained) for the i^{th} measurement between probe p and the first node of the j^{th} ASN on the path to target n. A local anomaly is then defined by

$$A_{p,n,i} = \forall j, i - h < j < i, \nexists H_{p,n,j} = H_{p,n,i} \vee \exists R_{p,n,i,j} > \left(\frac{1}{H} \sum_{h=1}^{H} R_{p,n,i-h,j} \right) \cdot \epsilon$$

where ϵ is a threshold for the deviation that occurs **K** consecutive times, **j** is the position of the value in the set vector and **H** is the last element of that same vector. This means that a local anomaly occurs when two of the following conditions apply: the path being compared does not exist in the historical set of the paths ($H_{p,n,i}$); when the path exists in the historical set, but the RTT value of one, or several, hops in the measurement being analysed is above the average value by a factor greater than ϵ. A global anomaly is given by

$$G_{n,i} = \left(\sum_{p} A_{p,n,i} \right) > \rho P$$

. This means that a global anomaly is declared when the number of probes in a local anomaly state are higher than ρP. It enters an anomaly state when the condition presented above is met, and will revert to the normal state whenever it no longer meets that condition.

Chapter 4

Architecture and Technical Design

This chapter aims to present a theoretical foundation for the solution implemented. This chapter will go through several topics to explain the solution and why it was chosen. The topics are as follow:

- Section **Requirements** layouts what the system must perform, and what is expected of it.
- Section **Architecture** presents the main system's architecture, components, communication and data flow.
- Section **Technical Design** describes the main components of the system and the design options.

Before going into a more detailed view of the architecture, some concepts have to be explained so that there is no confusion regarding the terms used to describe the platform. As previously explained, traffic on the internet flows from a source to a destination, going through several hops in between each other before reaching its final destination. For the purpose of explaining this platform, traffic sources will be identified as **probes** and traffic destinations as **targets**. To identify BGP routing anomalies, one must monitor the traffic flow from a probe to a given target or several targets. A BGP routing anomaly is a deviation from the standard path that packets usually take from a probe to a target. To help visualise that deviation, take a look at Figure 4.1. The flow represented by the green colour is what is classified as the normal flow from a probe to a target, and the flow in red depicts a routing anomaly from the same probe to the same target, flowing by a different path, to a terminal in between, which will be identified as a **relay**. In this context, a relay is a hop in the path that, in normal conditions, does not exist. By flowing through a relay, the path is altered, and that alteration of the path is what can be considered an anomaly. These deviations can happen for legitimate reasons, such as routing changes between ISPs, offloading traffic to less congested paths, etc. However, they may also be the result of malicious actions.
Two elements needed for the platform have already been identified: probes and targets. The targets, however, do not need any specific software, they just need to be reachable from the Internet. The probes will gather information regarding their respective targets, but there is a need for something else to glue it all together: a central server. This central server, which has been named as **Mainframe**, will control all existing probes, and receive information from

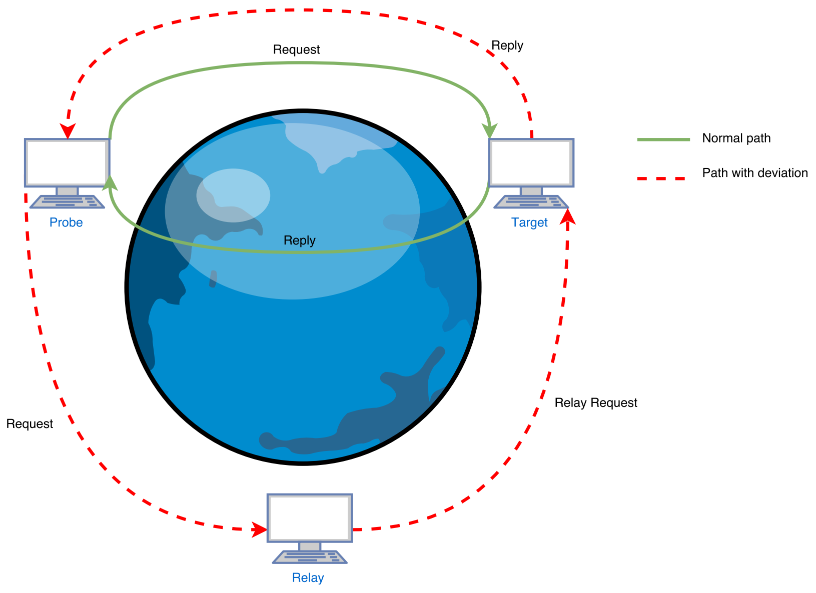


Figure 4.1: Traffic flow with and without anomaly

them. The two main components that are needed to be able to build this platform are then: probes (5.2) and a Mainframe (5.1).

4.1 Requirements

To create a solution for the problem expressed in this dissertation, there is a need to know what the system should be able to do (functional requirements) and what are the requirements to run the system itself (non-functional requirements). With that in mind, those two topics were separated and will be explained in further detail below. A more general architecture is depicted in Figure 4.2 to aid visualise the platform's elements.

4.1.1 Functional requirements

For this system to do what is proposed, it must obey these requirements:

1. There must be a way to add/remove probes and/or targets.
2. It must be able to configure new probes and control them in order to run the modules.
3. It must be able to add/remove/modify data gathering modules and data analysis modules.
4. It must be able to add/remove countermeasure scripts and assign them to targets.
5. It must be able to gather and store data as well as allow visualisation of said data.
6. Be able to utilise the data analysis modules to detect BGP routing attacks.
7. Notify when an attack is occurring.

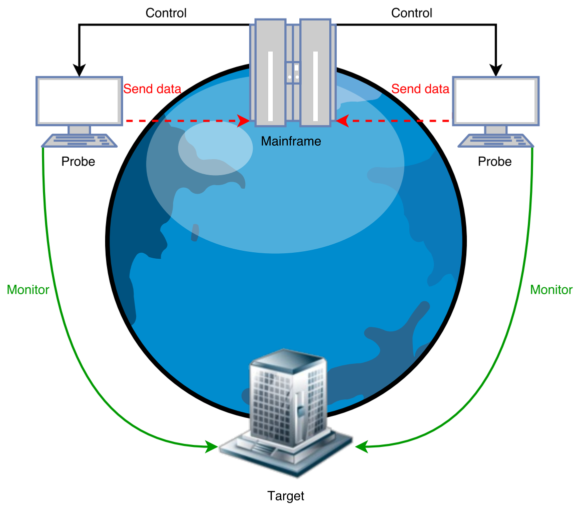


Figure 4.2: General architecture

8. Take action when an attack is occurring if countermeasures are active for that particular target.

9. Provide a graphical interface, in the form of a dashboard, for users to be able to control the system.

For the platform to be of any use, it must be able to monitor any given number of targets. To monitor the targets, the platform needs probes and a way to tell them what information they should gather. Raw data by itself is not of much use however and, as such, it must be handled and transformed into useful data and this is where the data analysis modules come into play; these modules use the raw data to give the system information regarding possible anomalies which in turn will be used to set off alarms and countermeasures. These are the wanted results from the requirements 1 through 8.
To make this platform easy to use and configure, it needs to have a graphical interface that runs on any terminal and provides all the required actions to control the platform and modify it; that is requirement 9. By providing a dashboard, which can be used from virtually any terminal with a web browser, it makes it easy to use the platform as long as one has connectivity to the server hosting the dashboard.

4.1.2 Non-functional requirements

For this system to be usable for the target audience, it must obey these requirements:

- Probe computational requirements must be low.
- It should be easy to deploy, with minimal dependencies required.
- It should be fault tolerant.
- Easy to maintain and modify.

One of the goals of this platform is to make it economically viable so that the maximum number of users can have access to it without a significant monetary cost. In order to do so, the requirements for the probes must be low, since they will mostly be hosted in Virtual Private Servers (VPS). Making this system easy to deploy and start up is also one of the goals; making it as easy as possible for the end user to set up this platform. Fault tolerance is an important point since the platform is meant to be monitoring several targets around the clock, it is imperative that it does not malfunction, or, at the very least, that it can recover from a malfunction without human interference unless it is a critical malfunction. Maintenance should be as simple as possible, and the platform will be open source so that if any modifications to the core system are desired, they can be done so with relative ease.

4.2 Architecture

This section aims to present how the components communicate with one another and what each component entails. The platform is composed of two major components: the probes, which run the **Central Probe Module (CPM)** and the central server **Mainframe**. For the platform to perform as desired, it requires the existence of several CPMs, spread around the globe, far away from the target destination.

As can be seen in Figure 4.3, the communications between both components happen in one of two ways: from the CPM, all communications are made using the provided Hypertext Transfer Protocol (HTTP)/Representational State Transfer (REST) Application Programming Interface (API) by the Mainframe. Mainframe to CPM communications are made using Secure Shell (SSH) and Secure Copy (SCP).

The CPM runs a docker container, which contains the main application. That container also has access to a shared folder in the file system, where all the active module's code is present, as well as all needed modules' configuration files and Rivest–Shamir–Adleman (RSA) keys.

The Mainframe application communicates with the file system, the database (MongoDB [42]), deploys the HTTP/REST APIs and Graphical User Interface (GUI) (Dashboard). The GUI also makes use of some APIs provided by the Mainframe's application.

4.3 Technical Design

As stated in the previous section, the implemented platform is composed of two major components, each one performs its own defined function for the overall functionality of the platform as a whole, but with very distinct roles and features. Those components are CPM and Mainframe; they are both implemented using Python language and the Python Flask framework. Communication between the various CPMs and Mainframe are possible via **Hypertext Transfer Protocol Secure (HTTPS)/REST** or HTTP/REST, and SSH. HTTPS/REST is used when the CPM is relaying information to the Mainframe and SSH is used for configuration of the CPMs by the Mainframe, as well as to invoke actions; the information flow between both components is secure, either by the use of SSH or with Advanced Encryption

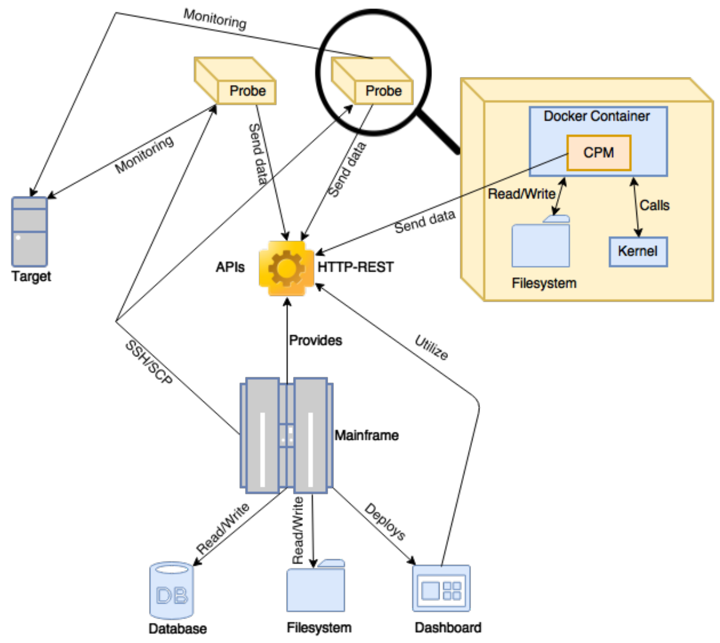


Figure 4.3: System architecture

Standard (AES) encryption and RSA signature. The security will be explored in more depth in the subsections below.

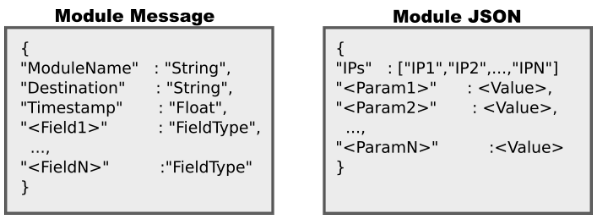
4.3.1 Central Probe Module

In order to be able to detect BGP anomalies to a given target T, one must be able to take measurements from a given probe to one, or several, targets. With that in mind, it becomes clear that one must create the needed tools to collect those measurements reliably and, as such, this component was created.

The function of this component is then to take measurements, utilising any given modules, and relay that information back to a central component: the Mainframe. To accomplish that and to allow the usage of one, or several, modules, the component's base code is built using Python Flask framework. The reason that Python Flask was chosen is that it allows for a lightweight platform that provides the needed tools for HTTP/REST API deployment as well as being a well supported and documented framework; with this framework, it was possible to dismiss concerns of resource competition from the different deployed modules, since its APIs endpoints already take care of possible concurrent accesses and, as such, one does not need to concern himself with preventing possible access collisions.

This component by itself does not take measurements or generate any type of data, but it supports the upload and usage of modules that can take such measurements; To do so, the uploaded modules must be python or python wrapped programs, and they must obey the following rules:

- The code must run on a *while True* cycle.



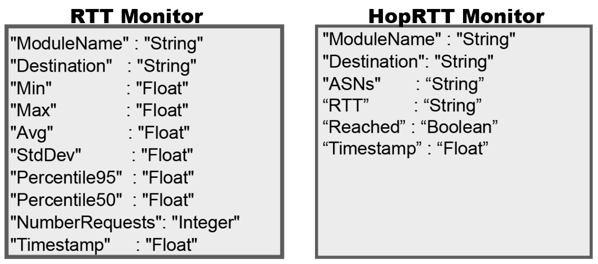
Module Message

```
{
"ModuleName"   : "String",
"Destination"    : "String",
"Timestamp"      : "Float",
"<Field1>"        : "FieldType",
  ...,
"<FieldN>"        :"FieldType"
}
```

Module JSON

```
{
"IPs"   : ["IP1","IP2",...,"IPN"]
"<Param1>"     : <Value>,
"<Param2>"      : <Value>,
  ...,
"<ParamN>"        :<Value>
}
```

Figure 4.4: Module message format (left) and configuration JSON file (right).



RTT Monitor

```
"ModuleName" : "String"
"Destination"   : "String"
"Min"              : "Float"
"Max"              : "Float"
"Avg"              : "Float"
"StdDev"          : "Float"
"Percentile95"  : "Float"
"Percentile50"  : "Float"
"NumberRequests": "Integer"
"Timestamp"     : "Float"
```

HopRTT Monitor

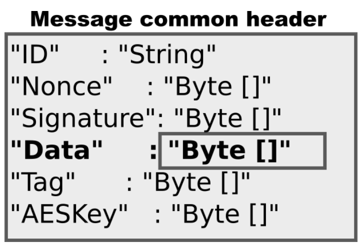
```
"ModuleName" : "String"
"Destination": "String"
"ASNs"      : "String"
"RTT"        : "String"
"Reached" : "Boolean"
"Timestamp" : "Float"
```

Figure 4.5: RTTMonitor fields (left) and HopRTTMonitor fields (right).

- It must read any needed parameters from a JSON configuration file named *moduleName*.json.
- Must support a list of target's IPs and iterate over them accordingly.
- Must submit all data with the depicted format in Figure 4.4 to the API endpoint on the CPM (*127.0.0.1:40000/receiveData*).

Whenever a new module is added to the CPM by the Mainframe, a restart is required on the CPM. The modules are invoked by the CPM itself and launched as independent threads, that only get terminated when the CPM itself is terminated. Currently, there are two modules build for the CPM: RTT Monitor and HopRTT Monitor; the former uses **ICMP** to send requests to the destination(s) and the latter performs traceroutes to the destination(s). A traceroute attempts to determinate all the hops (different network devices) through which traffic flows although this is not always possible, since nodes can be configured not to respond back to this kind of request and, as a result, some or all of the information may be missing. The fields for these two modules can be seen depicted in Figure 4.5. Whenever new data is sent from the modules to the CPM, that data is then packaged into a new message to be sent to the Mainframe.

Communication security is a concern when sending data that determines whether a BGP routing attack is occurring or not since manipulation of said data could result in false anomaly detection or the non-detection of ongoing attacks. To try and prevent that from occurring, the CPM contains its own RSA key pair, as well as the Mainframe's public RSA key. When sending a message to the Mainframe, the CPM generates a new AES key with which it encrypts the **Data** field, which is composed of all the fields in the module's message. That AES key is then encrypted with the Mainframe's public RSA key, and a signature is created using the CPM's private RSA key. The format of the message sent from the CPM to the Mainframe can be visualised in Figure 4.6. Each CPM is identified by a unique ID, generated on the setup by the Mainframe and that information is also sent in every message, in the



```
Message common header
"ID"      : "String"
"Nonce"    : "Byte []"
"Signature": "Byte []"
"Data"     : "Byte []"
"Tag"      : "Byte []"
"AESKey"   : "Byte []"
```

Figure 4.6: Message format sent from the CPM to the Mainframe.

ID field; this way, the Mainframe does not rely on the request's IP address but on that identification field instead.

4.3.2 Mainframe

Having data being gathered by several CPMs but having nowhere to store and analyse it would be pointless; as such, the Mainframe is the second and last major component of the platform. It is built using Python and Python Flask framework, as is the CPM. On top of that, it must be able to store a significant amount of data. Given the low relationships between data, a non-relational database was chosen: MongoDB. A relational database could have also been selected, but since relational databases must have predefined, non-dynamic schemas, it would pose an increased difficulty in the creation of a modular platform; non-relational databases are also most commonly used in big data applications, which this platform has the potential to be, depending on how many CPMs, targets and modules are involved.

This component is responsible for a number of elements that are crucial to the platform:

- Setup of new CPMs.
- Control and monitoring of the CPMs.
- Deployment of a GUI.
- API endpoints for the CPM.
- Receiving and using new modules, both Data Gathering and Data Analysis modules.
- Analyse data using the given modules and raise alarms when an anomaly is detected.
- Make use of uploaded countermeasures when an anomaly is detected.

Firstly, there is a need to explain how the setup of new probes is handled; when a new probe is added to the platform via the GUI, a new unique ID is created for that probe. The information regarding that probe is then stored in its respective collection. At this stage, the probe is added to the Mainframe, but no real action was performed yet; in order to setup the newly added probe, the user must select the action 'Setup' in the GUI and input the username and password for that probe. Since new packages will need to be installed, the user must have root access. Once this action has started, the Mainframe will attempt to login via SSH into the new probe, insert its public RSA key used for SSH in the probe's *authorized_keys* file, so that it does not need the password after the setup is complete. The next step is to configure the packages repositories and install Docker. After installation of Docker, a test is run to see if docker was successfully installed; if it succeeds, the next step is taken, else it will return an error message and abort the setup. The next step is to download the CPM image from the docker-hub repository and create a new container using that image; once downloaded and created, it will be run for the first time. The first run of the container will generate an RSA key pair for the new probe. Once the key generation is complete, the Mainframe will transfer the CPM's RSA public key and store it in a file named *probeID*.pub and also transfer its own RSA public key to the CPM. To finalise the setup, it will transfer two more files to the CPM: **mainframe.json** and **probeID.txt**. The former contains the Mainframe's public IP and Port where it is listening for requests while the latter contains the unique ID for the probe. This finishes the setup process and, if it is successful, the status of the probe will change accordingly in the database. The user's password used for this step is then discarded because storing it in the database would be a potential security risk.

The control of the CPMs is made using SSH and SSH only. From the GUI, it is possible to setup, start, restart and stop a CPM. When a CPM is started, the Mainframe will login into the CPM, this time without the need for a password, since its own RSA public key is already in the *authorized_key* file. It will then start the container, which will launch the CPM's main program and all the configured modules attached to it. The CPM will then start gathering data and send it to the Mainframe. The restart option simply shuts down and starts back up the container. If a stop action is done via the GUI, the container will be shut down, but the CPM will also be flagged as being offline by administrative action, and the Mainframe will not attempt to start it up again or active a possible a backup for the said CPM. When a CPM goes offline without administrative action, the Mainframe will try to start it up; if such is not possible, it will look for a backup for that probe and attempt to start it. If both those options fail, an alarm will be raised on the platform to notify the users that there is a probe that is no longer online. For the Mainframe to know which probes are online, each time a message is received from a CPM, the time stamp of that request is updated in the database. If a probe does not submit any message within a time frame less or equal to three times the lowest module cycle time, it enters an offline state and the actions mentioned above will be attempted.

The GUI also allows for the users to select which modules should be run on any given CPM, based on the target(s). For example, a CPM A with targets B and C can deploy ModA and ModB for target B and ModC for target C. When there is a change in targets or modules, the Mainframe will resend all needed module's code and configuration files using Secure Copy (SCP) and restart the container in order to reflect the changes.

The possibility to upload new modules is also present from within the GUI. There are two distinct types of modules: data gathering modules and data analysis modules. Data gathering modules take measurements and are used in the CPM; data analysis modules, on the other hand, are used in the Mainframe by accessing the data stored in the database to look for anomalies. Both of these modules must be implemented in python, or be python-wrapped, as that is the only way the code will be executed. There are a few rules that the modules must obey; for the data gathering modules, they are the following: each module must contain a JSON file with the fields depicted in Figure 4.7. If that requirement is not met, the mod-

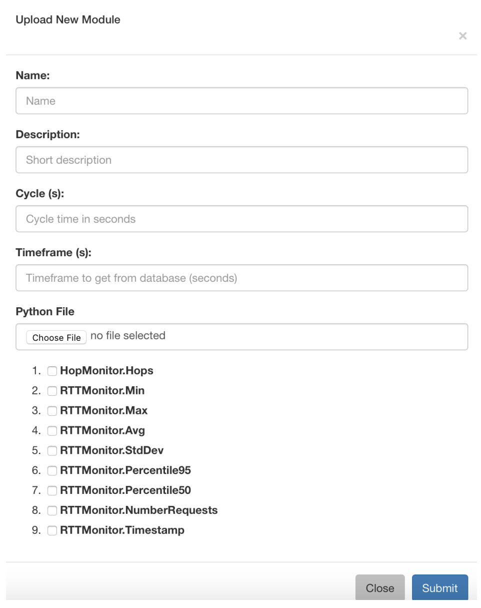
```
{
    "ModuleName": "String",
    "Params": {
        "<FieldName>": "<Value>",
        "<FieldName2>": "<Value>"
    },
    "Fields": {
        "Field1": "<Type>",
        "Field2": "<Type>"
    }
```

Figure 4.7: Data gathering module's JSON file required for upload.

ule will not be added to the platform. When uploading a data gathering module, it is also possible to upload a requirements file which includes the required dependencies for Python to be able to run the code. However, that is optional. As for data analysis modules, the only requirements are that the module accepts *-ip -file* flags when running the code. The *-ip* flag value is the target's IP address and the *-file* is the Tab Separated Values (TSV) file that contains the data; the return message must be a JSON string containing the field *Anomaly* which is a boolean (True or False), *Start* which is a float, *End* which is also a float, both values referring to when an anomaly started and stopped, and optionally a field named *Description*. When uploading a new data analysis module, the user must insert a cycle time (seconds), a timeframe (seconds) and select the wanted fields from the collections of the database. The cycle time refers to the interval on which the module will be run; the timeframe is how far back the data starts, for example, a timeframe of 1200 would retrieve the data for the last 20 minutes; and lastly, the selected fields will be the data that is written into the file. Figure 4.8 shows the form on the GUI for the data analysis module upload.

Another important feature is the ability to upload and deploy countermeasures scripts. When an anomaly is detected, an alert will be raised but there might not be anyone available to respond at the time, and if the information being sent and/or received is sensitive, one may want to take measures as soon as an anomaly is detected. Being so, the platform is able to run countermeasures scripts when an anomaly is detected to one, or several, target(s). After uploading a countermeasure script, the users can then select to which targets they want to apply that module; this module can also be uploaded together with a reset module, to be used after the anomaly has stopped, to reset any changes made by the countermeasure module although this is optional. The module should be written in Python, or Python wrapped, and must take a run argument with the flag *-ip*, in which the flag is the IP address of the affected target.

For CPMs to be able to relay information to the Mainframe, APIs are required. There is a single point of entry for all the CPMs to communicate with the Mainframe, which is located at *publicIPAddress:port*/receiveData. The data received here, as mentioned before, is AES encrypted. To decrypt the data, the Mainframe's private RSA key is used to retrieve the AES Key, and the CPM's public RSA key is used to verify the signature (this key was previously stored when the CPM was setup). If the signature cannot be validated or the data cannot be decrypted using the AES key, the data will be discarded; this prevents altered packets to be inserted into the database. When receiving a message from a CPM, its last activity timer is


Upload New Module

Name: Name

Description: Short description

Cycle (s): Cycle time in seconds

Timeframe (s): Timeframe to get from database (seconds)

Python File: Choose File no file selected

1. ☐ HopMonitor.Hops
2. ☐ RTTMonitor.Min
3. ☐ RTTMonitor.Max
4. ☐ RTTMonitor.Avg
5. ☐ RTTMonitor.StdDev
6. ☐ RTTMonitor.Percentile95
7. ☐ RTTMonitor.Percentile50
8. ☐ RTTMonitor.NumberRequests
9. ☐ RTTMonitor.Timestamp

Figure 4.8: Analysis upload form in the GUI.

set to the current time - this is done so that the CPM monitoring scripts on the Mainframe can detect offline CPMs.

It has already been referenced, but the Mainframe also implements the GUI. The GUI is where the users of the platform can control the whole system. There's a variety of actions that can be undertaken by them, and they are as follow:

- Add, delete and edit users and probes (CPMs).
- Add, delete and edit targets and respective modules for a given CPM.
- Add, delete and edit data gathering and data analysis modules.
- Add, delete and assign countermeasure modules.
- Start, restart, stop and setup probes.
- Visualise information regarding probes, modules, alarms, users and logs.

The actions above allow for a user to control the platform and make any necessary changes to the CPMs or any of the modules. It also allows for visualisation of information relative to the CPMs, log event in the Mainframe such as actions made upon probes and alarms.

Chapter 5

System Implementation

With the system architecture defined now begins the stage of implementing the platform. This chapter will dive in depth on how and why the different components of this platform were built, as well as explaining how all of the former come together to create a useful, working platform for this dissertation's objective. It is to note that all the code developed is open source under the MIT License. The chapter outline is as follows:

- Section **Mainframe** describes the components responsible for data storage, probe control, user interface, data analysis, anomaly detection, module upload and countermeasures.
- Section **Central Probe Module** describes the component responsible for gathering data from modules.
- Section **Data Gathering Modules** describes the currently implemented data gathering modules and how future modules can be uploaded and used.
- Section **Data Analysis Modules** describes the currently implemented data analysis modules and how future modules can be uploaded and used.
- Section **Countermeasures** describes the currently implemented countermeasure and how future ones can be uploaded and used.
- Section **Communication** describes how the different components communicate with each other, through the use of APIs or other protocols.
- Section **Graphical User Interface** describes how the user's interface was implemented, what is possible to do with it, and how can it be controlled by the users.
- Section **Deployment** describes how the deployment of the system was implemented.

5.1 Mainframe

This section aims to provide a clearer view of the implementation of the Mainframe, its internal organisation and what it provides for the platform.

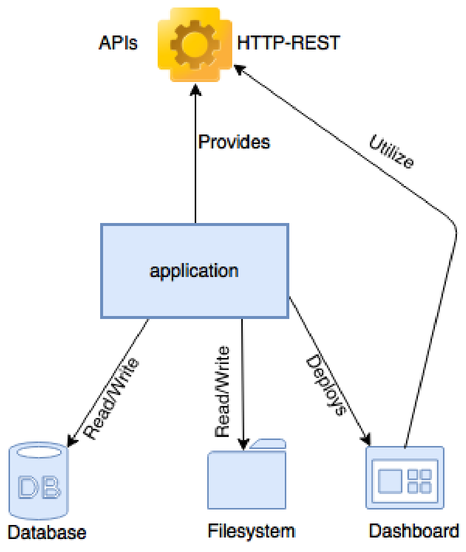


Figure 5.1: Mainframe internal organisation

5.1.1 Internal Organisation

This component is the core of the platform. It provides the means to receive and store data from several CPMs, manage and control probes, and provide the means for users to interact and control the platform. Firstly, one needs to know what the Mainframe contains within - Figure 5.1. As can be seen, the Mainframe is composed by the main application, which connects to a database (MongoDB) and several support code files and other documents in order to provide the APIs and deploy the dashboard (with all the functionalities required). The directory tree can be seen below:

```
.
|____common.py
|____flask_configs.txt
|____forms
|  |____forms.py
|  |____validators.py
|____json_to_mongo
|  |____convert_json.py
|____mainframe.json
|____mainframe.py
|____Models
|  |______init__.py
|  |____DataModels.py
|  |____Mod_HopMonitor.py
|  |____Mod_RTTMonitor.py
|____probe_scripts
|  |____misc.py
|  |____setup.py
|  |____start.py
|  |____stop.py
|____ProbesRSA
|  |___VVTAZLLV39A2G2CWJWXWYUEMA0Y0GEIY.pub
|  |___Y2TSXXTJAU1KH0LAWQ67H7IXYTV1ZQ7U.pub
|____RSAKey
|  |____key.priv
|  |____key.pub
|____static
|____templates
|____uploaded_analysis
|  |____monitor.py
|____uploaded_counter
|  |____monitor.py
|  |____test.py
|____uploaded_scripts
|  |____monitor.json
|  |____monitor.py
|  |____tracer.json
|  |____tracer.py
```

```
|____user_scripts
|  |____manageUsers.py
```

These will be further explained in the following subsections.

5.1.2 Application

This subsection aims to explain the implementation and the thought process behind the choices taken while implementing this component.

The application implements web-services using the Python Flask framework. It serves as the main application, which deploys all needed APIs for both the CPMs and the dashboard; it also communicates with the database and makes use of the several code files to perform the required actions for the platform. It was implemented using this framework due to the lightweight of it, as well as the ease of deployment. Inside the directory there is a configuration file, *flask_configs.txt*, for the application, which contains the *Host*, *Port*, *Pub*, *SSH*, *Database*, *dbuser*, *dbpass*, *dbhost* and *dbport*. These fields are necessary for the application to know on which IP address and port to run (Host and Port); where is the SSH key and the public RSA key used to access the probes, and allow communication and the information regarding the database name, IP address, port, user and password. Not only does this application deploy APIs, it also deploys several monitoring jobs on the platform itself: *checkProbes* will verify if any Probe is offline when it should not be and if that is the case, it will attempt to start it or start a backup if one is available; *checkProbesOffline* will check all probes to see if they have not communicated within a set timeframe and flag them as offline if that is the case. *checkThreadsAlive* is responsible for monitoring the data analysis modules and restart any thread that should be running but is not (due to errors or others), and finally *checkCounterMeasures* will verify if there is any active alarm that needs to be acted upon, such as starting a countermeasure or using the reset script if it exists and the alarm is no longer valid. These monitoring jobs are an important tool since, without them, the platform would have no usable way of detecting when probes are offline, if they need to be started, what alarms are active and deploy the assigned countermeasures. All of these jobs run on a timer, which means that even if one of them fails for some reason, it will be started up again at a later moment. This forms the basis of the platform: an application capable of reacting to events and deploy the needed actions if the events come to pass. Now let us break it up into smaller pieces and have a more insightful look at what composes this system and how it performs certain actions to accomplish the main goal: a modular platform, capable of providing the data necessary to detect BGP routing anomalies.

5.1.2.1 Database

This platform uses a Mongo database, which contains several collections; some are predefined while the remaining will be created when necessary. Inside the *Models* folder are the Python files that describe the collections. The *DataModels* file contains all predefined collections that exist in the platform's database. The remaining files are not predefined but are instead generated whenever a new data gathering module is uploaded to the platform. In order to do so, a script was created that reads from a structured JSON file and creates a new python file describing the collection as well as methods to insert documents and list all fields; the script's name is *convert_json.py*. In order for this script to execute correctly and create the needed object for the collection, the uploaded module JSON file must obey the structure depicted in Figure 4.7, which is: *ModuleName*, *Params*, *Fields*. The *ModuleName* must be a string and represents the name of the module; *Params* represents a list of parameters' name and value, which are used to create the module's configuration file; lastly, *Fields* represents a list of fields that must be represented in the newly created collection. The allowed types for the *Fields* section are : *int*, *String*, *Boolean*, *Long*, *Float* and *DateTime*. Here is an example of the output of this code generator script:

```python
from flask_mongoengine import MongoEngine
from datetime import datetime
db = MongoEngine()

class Mod_RTTMonitorList(db.EmbeddedDocument):
    Anomaly = db.BooleanField(default=False)
    Mod = db.StringField()

class Mod_RTTMonitor(db.Document):
    Source = db.StringField()
    Destination = db.StringField()
    Min = db.FloatField()
    Max = db.FloatField()
    Avg = db.FloatField()
    StdDev = db.FloatField()
    Percentile95 = db.FloatField()
    Percentile50 = db.FloatField()
    NumberRequests = db.IntField()
    Timestamp = db.FloatField()
    Anomalous = db.EmbeddedDocumentListField(Mod_RTTMonitorList)
    meta = {'collection':'Mod_RTTMonitor'}

    def addAnomaly(self,mod_Name):
        self.Anomalous.append(Mod_RTTMonitorList(Anomaly=True,Mod=mod_Name))
        self.save()

    def getAnomalyStatus(self,moduleName):
        if self.Anomalous == []:
            return False
        for anom in self.Anomalous:
            if anom.Mod == moduleName:
                return anom.Anomaly
        return False

def mod_addToRTTMonitor(data):
    try:
        new_obj = Mod_RTTMonitor(Source=data['Source'],Destination=data['Destination'],Min=data['Min'],Max=data['Max'],Avg=data['Avg'],StdDev=data['StdDev'],Percentile95=data['Percentile95'],Percentile50=data['Percentile50'],NumberRequests=data['NumberRequests'],Timestamp=data['Timestamp'])
        new_obj.save()
        return True
    except:
        return False

def mod_addAnomalyToInvervalRTTMonitor(start,stop,modName):
    objects = Mod_RTTMonitor.objects(Timestamp__gte=start,Timestamp__lte=stop)
    for ob in objects:
        ob.addAnomaly(modName)

def mod_listAllFieldsRTTMonitor():
    return ["Min","Max","Avg","StdDev","Percentile95","Percentile50","NumberRequests"]
```

To clarify, the script generates a new python file, under the *Models* folder, with the name *Mod_**ModuleName**.py* which contains the definition of the class that represents the collection, a method to add a new document to the collection and a method to list all the fields named *mod_addTo**ModuleName***, *mod_listAllField**ModuleName*** respectively. By using this script, it is possible to represent all new modules in the database as a collection as long as they obey the requirements of the JSON file. These are then used in the application to insert and retrieve data from the collections, using Python's *eval* method. It was necessary to allow for this platform to be truly modular, since if it did not exist, all data from new modules would have to be saved as a string in a preexisting collection, which would make the retrieval of the data harder and less efficient (parsing strings require more computation power than directly accessing fields).

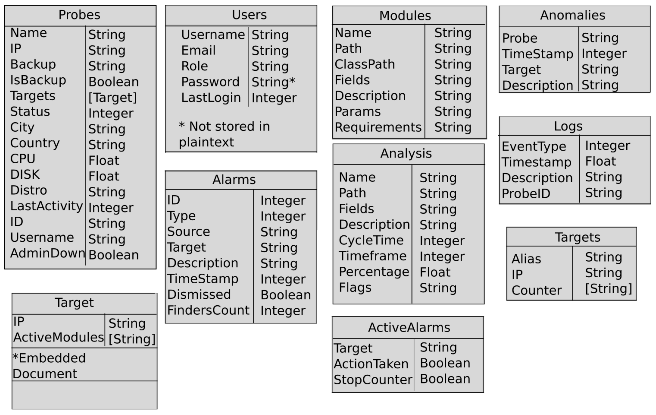


Figure 5.2: Database collections and respective fields.

Predefined collections have already been mentioned, but they have not yet been discussed. When developing the platform, there was a need to know what data to store and how to store it. With that in mind, the collections that are essential are:

- Probes - Stores the necessary information about the probes that exist on the platform.
- Logs - Stores logs from probes such as setup, start, restart and stop actions.
- Targets - Stores information regarding the targets' details and active countermeasures scripts.
- Counters - Stores information regarding the countermeasures scripts.
- Users - Stores information regarding the users of the platform.
- Modules - Stores information regarding the data gathering modules.
- Anomalies - Stores information regarding detected anomalies.
- Alarms - Stores information regarding present and past alarms.
- ActiveAlarms - Stores information regarding currently active alarms and if measures have been taken.
- Analysis - Stores information regarding the data analysis modules.

The fields of the collections can be seen in Figure 5.2.

5.1.2.2 Probe Management Scripts

This subsection will explain what scripts exist, their function and how they have been implemented.

In order to control the probes, the Mainframe needs to be able to access them via SSH and send any needed files via SCP. For that purpose, four python scripts have been created

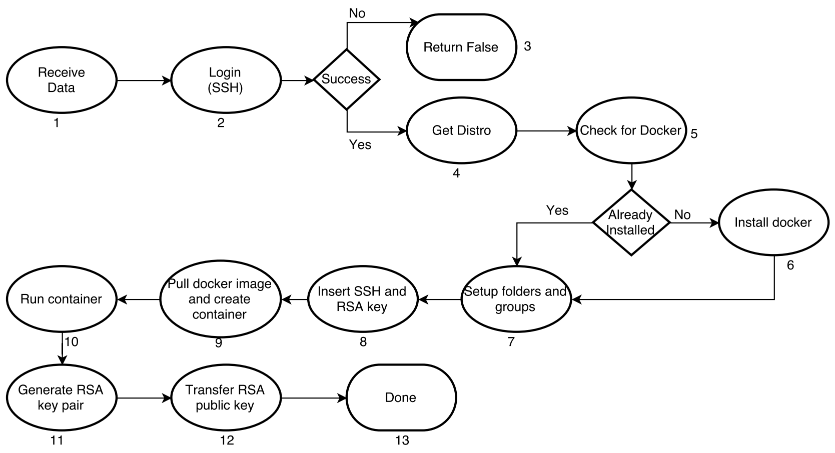


Figure 5.3: Setup script flow.

to facilitate control: setup, start (doubles as restart), stop and another script named *misc* which is used to reflect modules changes (removing, adding or altering). All these scripts are executed by creating and launching a new thread each time they are called. To understand these scripts easily, these will be explored with the aid of diagrams. Let us start with the setup script.

When the setup script is executed from the Mainframe, the flow is the one depicted in Figure 5.3. On step 1, it receives all required data to be able to setup the new probe: login username and password, IP, mainframe's public RSA and SSH key paths, probe ID and a JSON file which contains the Mainframe's public IP address and the port on which it is listening. It will then move to step 2 and try to log in; if that fails, the script will exit with a value of *False*, else it will try to get the Linux Distro name (Debian, Ubuntu and CentOS are supported). In step 5, the script verifies if docker-ce is already installed and, if so, proceeds to step 7; if however, it is not installed, it will execute step 6 and install docker-ce. After docker is installed, the next step is to setup the folders and assign the user to the docker group so that it can run containers without the need for root privileges. Step 8 will insert the Mainframe's SSH key into the *authorized_keys* file for SSH on the probe and also upload its RSA public key into the previously created folder. After that is done, it is now time to download and create a new container, which is done in step 9. In order to know if the container is running for the first time, a script was made to check for the RSA keys; if they are not found, then the script considers this is the first run, generates the keys and terminates the container. The script will then wait on step 11 until the RSA key pair has been created and finally, it will transfer the probe's RSA public key to the Mainframe. That completes the first script, and from this point on, the probe is ready to be used.

The next script is the start script, which also doubles as restart. Let us take a look (Figure 5.4). It starts the same way as the setup script, but this time the only needed information is the username and the IP address. From there, it will attempt to log in via SSH (step 2), stop the container (if it is not running, it will simply fail the stop command) and start it back again. It will then verify if the container is indeed running and return a positive or negative answer, *True* or *False*, respectively.

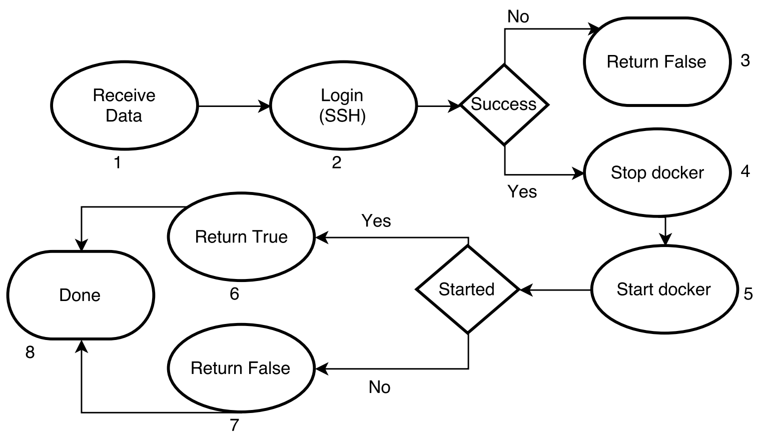


Figure 5.4: Start script flow.

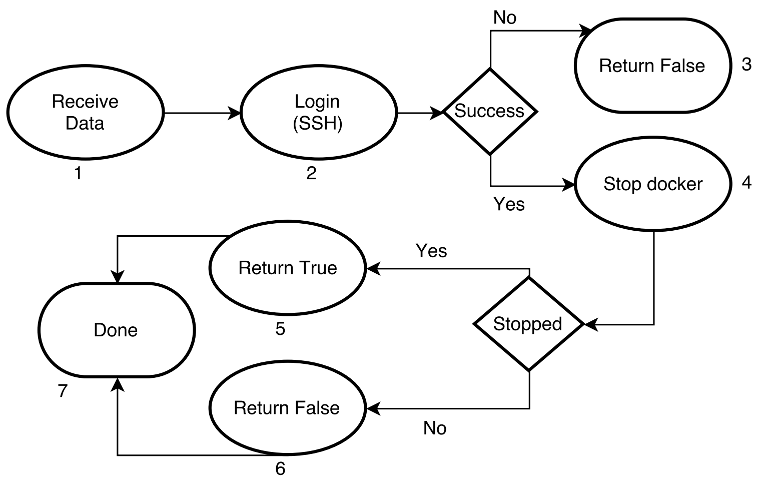


Figure 5.5: Stop script flow.

The stop script (Figure 5.5) behaves similarly, containing the same steps, with the exception of step 5, which does not exist.

The next, and last, script is used to transfer the modules' files so that the changes are reflected in the probe. Figure 5.6 depicts how it works.

This script requires more data than the start and stop scripts; besides the username and IP address, it also requires a list to the path of the modules to add, a list of configuration parameters for those modules and a list with the configuration files' name. After logging in, it will delete all files inside the probe's modules folder and transfer the files. If one, or more, requirements files (for pip) exist on the modules' path, they get concatenated into one and transferred to the probe as well, to be installed on the next container run. After the file transfer is done, it will create the configuration files (JSON) to be used by each module, transfer them and then exit. This script does not perform any action on the container itself, but the Mainframe makes sure the new changes are reflected by restarting the container after the script is done executing.

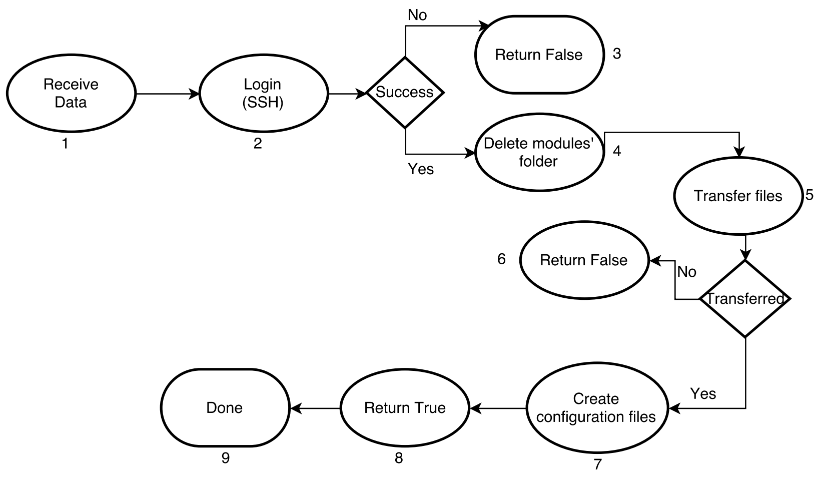


Figure 5.6: Misc script flow.

These scripts are paramount to the proper functioning of the Mainframe. Without them, the Mainframe would not be able to control the probes directly and would have to rely on other approaches to control them. Since SSH and SCP are standard in most, if not all, Linux distributions, the author decided to take the most advantage possible of those tools in order to implement the platform's control methods.

5.2 Central Probe Module

This section will revolve around the implementation of the CPM. It will be explained how and why this component was implemented, its features and its internal organisation.

5.2.1 Internal organisation

To understand this component, its internal organisation and several components need to be known. They can be visualised in Figure 5.7. Let us take a closer look at the docker container and the file system organisation.

5.2.1.1 Docker container

For deployment to be as smooth as possible and not prone to failure due to different Unix system configurations, Docker was chosen to be used as a container. A container allows this platform to be deployed without concerns about dependencies, libraries or other issues that could arise from different configurations; by wrapping every needed component in a container, which takes care of communication with the operative system's kernel, one can mitigate the effects of running the platform on different systems. Docker makes it so that it is possible to simply create a container from an existing image in the desired machine, and have it up and running in a matter of minutes.

The docker image being used was created with all the needed code files and dependencies

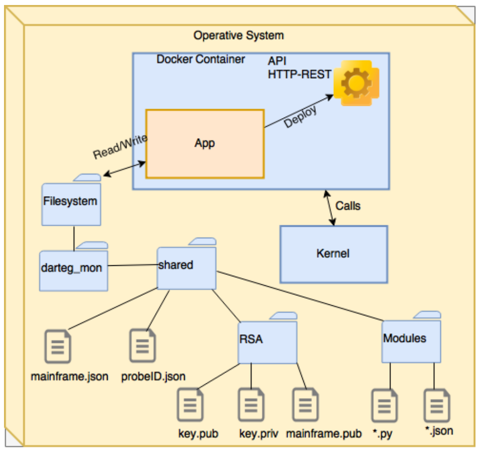


Figure 5.7: CPM internal organisation

(running Ubuntu 16.04 as the base system); it was then uploaded to a repository, from where the image can be downloaded and the container generated. This image contains server.py (application code) and Python scripts to check for RSA keys, and generate them if they are missing. When starting up, it will also check for new Python dependencies that may be required by looking for a requirements file in the shared folder and installing it if one exists. The Dockerfile can be seen below:

```
FROM ubuntu:latest
RUN apt-get update -y
RUN apt-get install -y python-pip python-dev build-essential iputils-ping
    iputils-traceroute
COPY . /app
WORKDIR /app
RUN pip install -r requirements.txt
EXPOSE 40000
CMD pip install -r shared/Modules/requirements.txt; if python checkRSAKeys.py;
     then echo "setup done"; else python server.py; fi
```

When this container is created, the shared folder */darteg_mon/shared/* is mounted to the container, and it is remounted every time the container starts. Let us then take a closer look at the contents of that folder in the next subsection.

5.2.1.2 Shared folder

It is now known how the CPM application is present in the system, but there is still a need to know what else is required in order for it to function properly. This is where the shared folder comes into play. This folder contains:

- mainframe.json - Contains the IP address and Port of the Mainframe's APIs as well as the information regarding if HTTPS is enabled.
- probeID.json - Contains the unique identifier of this probe.
- RSA folder - Contains the probe's RSA private and public key, as well as the Mainframe's RSA public key.

- Modules folder - Contains all the active modules to be used, together with their respective JSON configuration files.

This folder and its contents are created when a new probe is initiated; before starting the container, the directory structure is created, *mainframe.json, mainframe.pub and probeID.json* are uploaded into the probe. After doing so, the docker image is downloaded, and a new container created. When that container runs for the first time, it will generate the RSA key pair for the probe, with 4096 bits. This key will then be downloaded by the Mainframe.

5.2.2 Application

In order to gather data, an application capable of supporting modules that generate measurements was needed. This is what this application does at its core: provides a way for those modules to run as well as providing an API to where the data can be sent. By making use of the internal organisation, the application launches all present modules in the shared folder, under the directory *~/darteg_mon/shared/Modules*. It does so by iterating over the folder, finding all files with a Python extension (*.py) and launching a thread that runs the module until the application itself terminates.

```python
def getModules():
    subdirs = [subdir for subdir in os.listdir(os.getcwd()+'/shared/Modules/')
     if os.path.isdir(os.getcwd()+'/shared/Modules/'+subdir)]
    files = os.listdir(os.getcwd()+"/shared/Modules/")
    tmp = []
    for subdir in subdirs:
        files = os.listdir(os.getcwd()+"/shared/Modules/"+subdir)
        for f in files:
            if str(f).endswith('.py'):
                tmp.append(subdir+'/'+str(f))
    return tmp

def threadFunction(moduleName):
    try:
        subprocess.check_output("python " + os.getcwd() + "/shared/Modules/"+
    moduleName, shell=True)
    except subprocess.CalledProcessError:
        print("Failure to initialize module")

def startModules():
    threadList = []
    try:
        for module in getModules():
            threadList.append(threading.Thread(target=threadFunction,args=(
    module,)))
            threadList[-1].daemon = True
            threadList[-1].start()
    except:
        pass
```

These modules will then start generating measurements and send them to the CPM's API located at **localhost:40000/receiveData**. Upon receiving this data, the CPM will create a new message with the required fields and send it to the Mainframe via the provided API. The modules could have been made to run without the using of the CPM, but that would require that each and every single module implemented all the needed communication features currently present in the CPM and the Mainframe, making the overall modular experience less user-friendly. By providing a single point of access for all modules, the requirements for new modules are less extensive than if done otherwise. The CPM then handles the launch of all the active modules, as well as taking care of the communication with the Mainframe, in a way that is simple and fast to deploy.

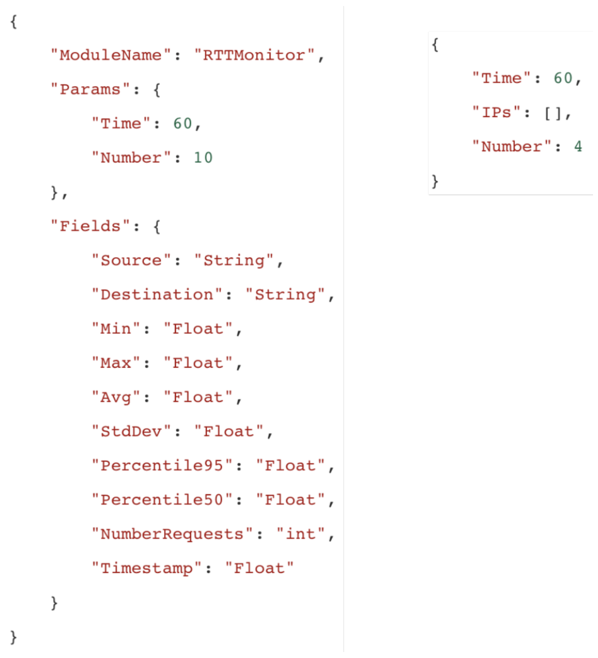
5.3 Data Gathering Modules

This section will explain what is the purpose of these modules, how they can be implemented in order to be accepted by the platform and the ones that have been already implemented will be explained. As previously stated, the platform is capable of receiving new code and use it in the form of modules. There are restrictions, however: these modules must obey certain requirements. The main requirement is that the code must be written in Python or with a Python wrapper. The rest of the requirements vary depending on the type of module being uploaded. Since this section is dedicated to the data gathering modules, it will elaborate on those only.

These modules are responsible for generating data and metrics to be used by the data analysis modules in the platform, in order to detect possible anomalies. Since that is the case, these modules will then be used in the CPMs. In order to be accepted into the platform and be used by the CPMs, they must obey the following rules: use a JSON file with the configurations regarding any needed parameters (IP address list is mandatory); run in an infinite while loop; iterate over all IP addresses in the list; relay all information to the CPM via the API located at *localhost:40000/receiveData*, respecting the message format shown in Figure 4.5. If this format is not respected, the delivery of the data to the Mainframe will fail, rendering the module useless.

For the modules to be used, they must first be uploaded. When uploading a new module, a new collection in the database will be created for it, alongside with three methods to insert new documents, list all the collection's fields and flag documents as anomalous (as can be seen in 5.1.2.1), all of these defined in a new Python file. Due to that, and since it is not possible to reload imports at run-time in Python, a restart to the mainframe must be done before they can be used. All new modules will have the following mandatory fields (that can be omitted from the JSON upload file): *Source*, *Destination*,*Timestamp* and *Anomalous.* These are used to know from which probe the data came from (*Source*), the destination (*Destination*), when was that data obtained (*Timestamp*) and to know if the document was flagged as being a possible anomaly (*Anomalous*) - this field may or may not be altered, depending on the data analysis modules' output.
To upload a new module, the code, configuration file and all needed support files must be submitted as a *.zip* file, as well as a separate JSON file with the format depicted in Figure 4.7. The zip file must obey the following structure:

```
.
|____configurationfile.json (mandatory)
|____folders (optional)
```



```
{
    "ModuleName": "RTTMonitor",
    "Params": {
        "Time": 60,
        "Number": 10
    },
    "Fields": {
        "Source": "String",
        "Destination": "String",
        "Min": "Float",
        "Max": "Float",
        "Avg": "Float",
        "StdDev": "Float",
        "Percentile95": "Float",
        "Percentile50": "Float",
        "NumberRequests": "int",
        "Timestamp": "Float"
    }
}
```

```
{
    "Time": 60,
    "IPs": [],
    "Number": 4
}
```

Figure 5.8: JSON fields file (left) and JSON configuration file (right)

```
|____main_file.py (mandatory)
|____requirements.txt (optional)
```

It can have any number of folders, as long as the files marked as mandatory are in the root folder (and the requirements.txt too, in case it exists). When this file is uploaded, the Mainframe will check if the mandatory files are present, and if so, create the collection and methods previously mentioned; after that is done, the folder and all its files will be moved to the *uploaded_scripts/* folder, within a folder named after the module. A new document will also be inserted into the collection *Modules*, with all the parameters that can be seen in Figure 5.2. After a restart to the Mainframe, the new module will be ready to be used and deployed to the selected probes.
The ability to do this allows for users to create, or improve, modules for a more flexible and modular experience.

5.3.1 Implemented Modules

This subsection aims to provide a more in depth view of the already implemented data gathering modules: RTTMonitor and RTTHopMonitor. Let us start with the former.

5.3.1.1 RTTMonitor

This module is responsible for monitoring the RTT to one, or several, targets. As previously mentioned, it does so by sending ICMP requests to the destination(s) and retrieving the RTT from those requests. One can choose how many requests to do per iteration on a target, by simply changing the parameters on the dashboard. However, it is advisable to send at least 10 requests per iteration, to try and minimise skewness of the data. The data obtained by this module is then used to create metrics: minimum, maximum, average, standard deviation, percentile 50 and percentile 95. These are the metrics that will be sent to the Mainframe and stored in the database, to be used by the data analysis modules. The goal of this module is to provide information related to the delay of the communications from a probe to one or several targets, in order to provide enough data to detect any significant changes in those values and attempt to extract information from that. This module, as all other data gathering modules, follows the same requirements as the other ones. As such, the configuration file and the JSON upload file were created as well. They can be seen Figure 5.8. The zipped folder for upload contains the code (RTTMonitor.py), the configuration JSON file (RTTMonitor.json) and the requirements file (requirements.txt).

5.3.2 RTTHopMonitor

This module is responsible for monitoring the path to one, or several, targets. The objective is to be able to provide enough data so that it can be used to pinpoint the hop where the anomaly started. When traffic flows on the internet, it may go through one or several different ASNs. It is then possible to capture the ASNs through which traffic flows and the respective RTT it takes from one to another. This module then provides the following data regarding the path from source A to destination B: number of hops, RTT between hops, ASN of each hop, IP of each hop and the name assigned to it (if any), and lastly if it could reach the destination or not (some hops don't answer back, which may cause lack of information). With this information, one can try to pinpoint where the anomaly started.

5.4 Data Analysis Modules

Mod1.Timestamp	Mod1.Anom	Mod1.Field1	Mod1.Field2	Mod2.Timestamp	Mod2.Anom	Mod2.Field1
1501155649	False	30.5	5	1501155630	False	12.1
1501155679	False	28.9	7	1501155640	False	14.2
1501155709	True	120.2	8	1501155650	False	13.8
...	...	...	...	...	...	...
-	-	-	-	-	-	-

Table 5.1: Input data file sample for analysis modules.

This section will explain what is the purpose of these modules, how they can be implemented in order to be accepted by the platform and the ones that have already been implemented will be explained. As with the data gathering modules, these must also obey certain rules in order to be accepted by the platform. The rules for these are less strict than for their counterpart since all that is needed is a zip file containing the code with the same folder organisation of the data gathering modules. These modules are responsible for utilising the data that is collected by the data gathering modules, which is available in the database. Since these modules will not have direct access to the database collections themselves, because that would require the users to know beforehand how the database is organised and all the fields contained within, but also to prevent unwanted operations over the information stored, these modules will be able to select the fields that they require. This selection is done via the dashboard, and the fields available will be present as ModuleName.Field. Other than the fields selected via the dashboard, two more fields will always be available for the module: Timestamp and Anom. The Timestamp field contains the information regarding when that data was collected, and the Anom field provides the means to know if this entry has already been classified as anomalous by the module being used. Both these fields will also have the respective ModuleName associated (the ModuleName refers to the data gathering module that obtained the information). Given that the modules do not have direct access to the database, the Mainframe will create a TSV file contain all the selected fields, organised from most recent to less recent, based on the Timestamp, which will then be passed as an argument to the respective data analysis module. Since the file is passed as an argument, these modules must accept the execution flag *-f* and handle the reading of the file themselves. Failure to do so will result in the data gathering module to not receive any data. More flags can exist, and these can be changed in the dashboard. To help visualise the output of these modules, let us take a look at Table 5.1. As can be seen, the file will contain all the selected values with the addition of the *Timestamp* and *Anom* fields. The order of the fields will be the same as in the first line of the file, which will be the header and contain the respective names of the fields. With that in mind, the module should be coded in a way that it can parse the header to know in which positions the values are. How far back the received data goes depends on the value of the *Timeframe* selected for the module on the dashboard. What this does is, for example, with a Timeframe of 3600 (seconds), the output TSV file will contain all values from the moment of execution (T0) all the way back to $T0 - Timeframe$.

Now that it is already known what the input is let us talk about what the module must return. The caller of these modules expects that a JSON formatted string is returned to the terminal (in this case, printed). The fields are mandatory and will result in module failure if not met. The fields are: *Anomaly* - boolean, *Start* - float (Timestamp) and *LastStamp* - float (Timestamp). The first field is self-explanatory, it will be true if an anomaly has been detected or false otherwise; the two following fields mark the beginning and end of the anomaly detected so that the respective data can be flagged as anomalous in the database. This flagging is always done regarding the data analysis module that flagged the data and not globally, i.e., if moduleN flags the data as anomalous, moduleY will still receive the data as not anomalous since it was not that module that flagged it.

5.4.1 RTTAnalyzer

As part of the work developed, a data analysis module was also created. It is named *RTTAnalyzer* and makes use of the data from the data gathering module *RTTMonitor*, implementing the methodology described in 3.2.1. This module has several variables that can be changed via the dashboard. These variables are passed as execution arguments and are the following: *threshold*, *delta*, *count* and *maxOccur*. The threshold represents how much higher the RTT value can be, compared to the historical data, before it is considered anomalous; delta is the time frame (in seconds) in which to segment the data received; count is the minimum number of values that can be used to calculate the average of the previous data and maxOccur is the number of anomalous occurrences in a row before considering it to be an anomaly. The usage of maxOccur is there to prevent the event of, for example, in 20 values, only one of them being anomalous while the remaining are not; if the module considered an anomaly to exist as soon as a single value was considered anomalous, the number of false alarms would most likely be high, since it is not unusual for an abnormal RTT value to appear due to either equipment overload or some other issues. To better understand how the data is processed, Figure 5.9

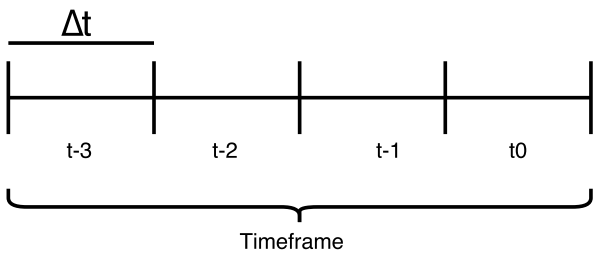


Figure 5.9: Segmentation of data.

shows the way that data is segmented by this module. The module will start by retrieving the first two delta intervals t0 and t-1. T0 will contain the values to analyse, and t-1 will provide the values to which it will compare the former against. To compare the values, first it must compute the average of t-1. The module will not take into account any values flagged as anomalous, and if the number of values remaining it is too low (below the count variable), it will then attempt to calculate t-2, t-3, etc., up until it can calculate an average or runs out of data. In the event it runs out of data before being able to calculate an average, it will assume that the values flagged as an anomaly are now the new normal and calculate t-1. After the average is obtained, each of the values in t0 will be compared to the average, being flagged as anomalous if they exceed the threshold. After K consecutive anomalous values, K being the value of the variable maxOccur, the module will enter an anomaly state, and return that information to the Mainframe.

5.4.2 RTTHopAnalyzer

To obtain valuable information from the data gathered by the monitoring module RTTHopMonitor, an analysis module was created. This module implements the methodology described in 3.2.2. Similarly to RTTAnalyzer, this module has several variables that can be changed via the dashboard. These variables are passed as execution arguments and are the following: *threshold*, *delta*, *count* and *maxOccur*. All of these arguments have already been explained for the RTTAnalyzer module. This module implements the methodology by receiving the data from the dashboard, using it to create a historical data set of the path vectors taken in the path, as well as the respective RTTs for each of the hops, and uses the dataset as the base for the comparison of the new data. This historical data set is obtained by the same process applied by the other analysis module. If the path vector being analysed does not match any of the previously observed vectors for K consecutive times, it is flagged as an anomaly, and that information is returned to the Mainframe. However, if the vector does match the previously observed vectors, it will match the vector being analysed with the previously observed ones, and obtain the average RTT value for each of the hops in the corresponding vector. It will then compare the RTT values for each hop. If the RTT data currently under analysis goes beyond the threshold ϵ, K consecutive times, it will be flagged as an anomaly, and that information will be returned to the Mainframe.

5.5 Countermeasures

Knowing when an attack is happening may not be enough. One may want to act upon it as best one can and deploy some measures to mitigate the risks to themselves. This is where the ability to add and assign countermeasures to a given target, or several, comes into play. A countermeasure is an action that will try to counteract a certain event. For example: if a network's traffic is currently being deviated to abnormal paths, one could want to stop the flow of information to that network, or increase the communication's security level by increasing encryption or by other means. The ability to do this in real time, i.e., when an anomaly is detected, can provide more secure operations for the user deploying them, by deploying the measures as soon as the attack is detected. These measures can be uploaded and assigned to any given target(s) via the dashboard as a .zip file, as well as a script to revert the changes made by the countermeasures after the attack is over. Both scripts must accept *-ip* as an execution argument, in which the IP address of the Target for which an anomaly was detected is passed. By doing this, not only does the user know when the attack happened, but he can also act upon it by setting these measures in place when first setting up the target(s) on the platform. These measures can be deployed at a service level, such as web servers, at a network level by changing routes, denying traffic, blocking ports or others. This allows for better protection since one can deny outbound, or inbound, traffic whenever an anomaly is detected. However, these actions may also come at the cost of connectivity loss, unreachable services for end users, added overhead for communications (when encryption levels are increased), etc. As such, one must carefully weigh the pros and cons of these measures. Is data security of extreme importance for all services or just for some? Does it make sense to completely sever all communications or is it better to deploy less extreme measures? These questions should be answered when deciding if it is worth it to deploy a countermeasure. If the data is sensitive, but the operations must go on, then perhaps increasing the levels of encryption is a better measure than shutting all communications. On the other hand, if the services being used are not essential, it might be better to just completely shut them down instead of increasing the encryption levels and, therefore, increase the load of the networking equipment and/or servers. Since the users of this platform may have different goals regarding what to do when an anomaly is detected, the option is there to act if one does wish so. Two different measures also exist on the platform, as will be explained below.

There are currently two implemented countermeasures: one meant to be used in a web server running Apache 2 and another to be used on a firewall running on a Linux machine with iptables active. The first measure, named SiteSwap, logs into a machine running Apache 2 via SSH and disables the current website and activates another one displaying an error message. The before and after can be seen in Figure 5.11. This script works by utilising Apache 2's commands to enable and disable a website that is running on Apache 2. By doing this, one can prevent users' information to be potentially stolen while utilising the website, since the website active during an anomaly will only be displaying an alert or error message and will not allow for input of any type of information. The second implemented measure, named BlockVLAN, acts upon a, or several, Linux machine(s) running a firewall with iptables. In the same way as the previous measure, it logs in via SSH to the machine and performs actions on that machine's iptables. This measure in particular blocks all known Voice over Internet Protocol (VoIP) and video vlans from communicating with the destination IP address that was passed as an argument to the script (where the anomaly was detected). It is done by using the iptables' commands by telling it to drop any packets coming from one,

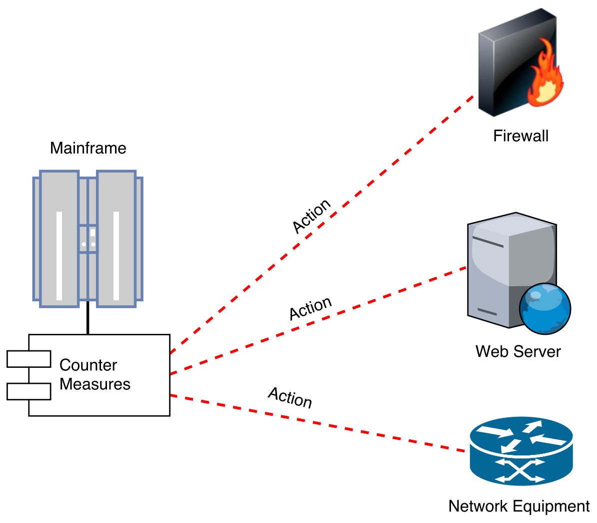
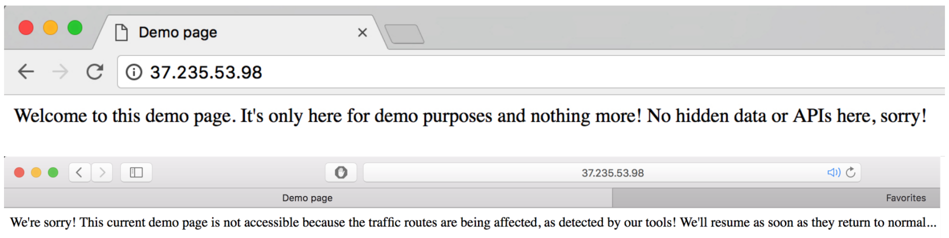


Figure 5.10: Countermeasures examples



Welcome to this demo page. It's only here for demo purposes and nothing more! No hidden data or APIs here, sorry!

We're sorry! This current demo page is not accessible because the traffic routes are being affected, as detected by our tools! We'll resume as soon as they return to normal...

Figure 5.11: Normal website (top) and error website (bottom)

or several, network(s) that have as destination the targetIP, essentially blocking all outbound communications from those networks to the target destination. Both of these measures also contain a reset script which will do the opposite and, by doing so, restore the normal working configurations. The first measure aims to protect the services' users by letting them know that something is wrong while simultaneously not allowing users' information to be sent and the second one attempts to protect communications to the exterior by not allowing them to happen in the first place.

5.6 Communication

This section aims to elaborate further on the communication between the several components, including the dashboard, as well as the security measures in place to minimise the risk of data theft or data injection (such as crafted packets).

5.6.1 Users to Mainframe

As stated previously, the communication between components (CPM and Mainframe) is established via HTTP/REST or HTTPS/REST APIs when the communication is done from the CPM to the Mainframe or via SSH and SCP (for file transfer) when the communication

is done the other way around (Mainframe to CPM). When using SSH and SCP, the security of the connection is already in place due to the usage of those protocols but, when the communication is done between the CPM to the Mainframe, the security provided by HTTPS may not be enough since Secure Sockets Layer (SSL) strip attacks are possible as are attacks that involve modifying the certificate [43] . Not only that but HTTPS would only provide data encryption with no means of authenticating the senders, and, as such, another layer of security on top of what HTTPS already provides was added, while adding sender authentication. The reason for implementing authentication is to prevent unauthorised data packets (originating from others than the legitimate probes) to be inserted into the database.
Regarding the access to the dashboard by the users, the credentials used to log into the dashboard are protected using HTTPS, by utilising a **Certificate Authority (CA)** certificate, that is already present in the Mainframe (would need to be manually added to the browser, else it would state that the access is not secure), or by inserting a valid certificate from a number of different trusted CAs.

5.6.2 CPM to Mainframe

As stated in the subsection above, it was decided to implement the security by other means other than HTTPS. To do so, two different algorithms were used: **RSA** and **AES**. By combining these two algorithms, one can achieve two things: encryption of data and sender authentication. Let us start by taking a look at how this was done with the RSA algorithm.

RSA, unlike AES, does not use a single key. In fact, RSA uses two keys: a private key and a public key. The private key should never leave the original's owner terminal, while the public key can be distributed to anyone. This means that two different terminals can communicate securely, as long as both of them have their own private key and the destination's public key. The traffic sent to the Mainframe is encrypted using the receiver's public key, and decrypted using the receiver's private key (which only he has); as such, communications should be secure. However, RSA can only encrypt up to the number of bytes of the key, so a key with 4096 bits (which is the one in use) would only be able to encrypt data up until 512 bytes, including the padding and headers, and that may not be enough, which is why AES is used together with RSA (this limitation could be lifted if encryption was done on a block basis, but the common practice is to pair RSA with symmetric encryption processes). Another reason for RSA is that it also provides the means to authenticate the sender, by using its private key to generate a hash and encrypting it with its private key; this way, when a message is received, the inverse process can be done, using the public key of the sender, and if the hashes are equal, the signature is valid, and the sender is authenticated.

So, now that the communications are signed, the Mainframe can confirm if a message comes from a legitimate source or not (discarding any that it cannot verify the signature to). However, the need to encrypt the data itself is still present. Having that data 'floating' around on the internet without any encryption is not the best case scenario and, as such, that needs to be addressed. In order to resolve that issue, the use of AES encryption was chosen. AES is able to handle data of arbitrary length, being a suitable match for the needs of the communication. In this instance, AES-256 bit was used. AES-128 could have been used in-

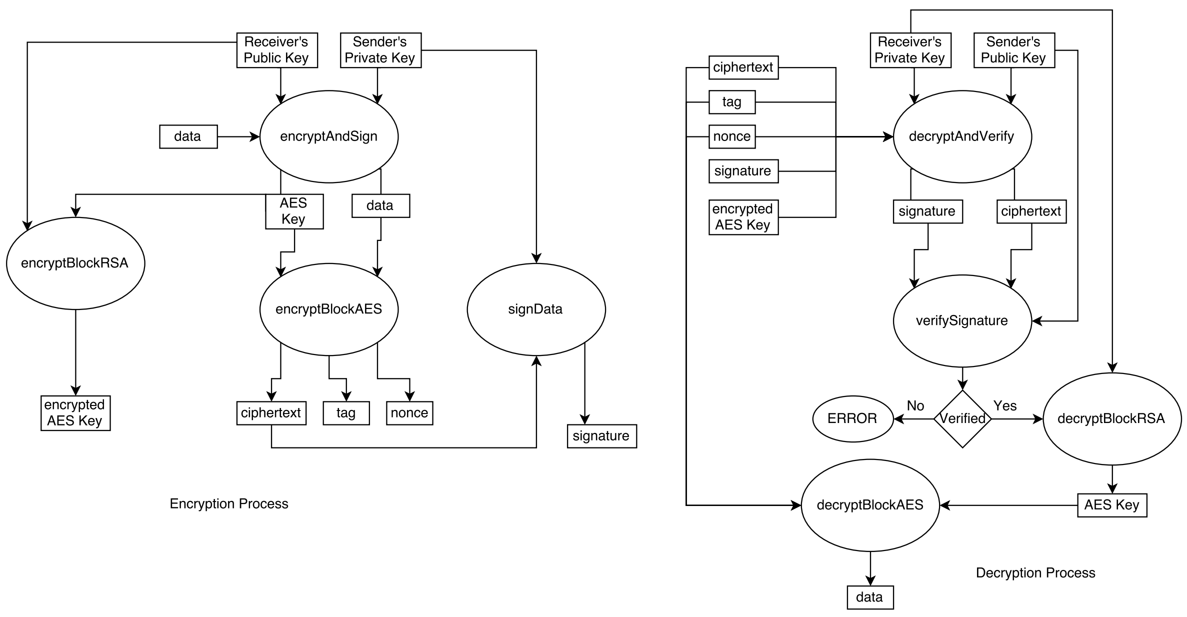


Figure 5.12: Encryption (left) and decryption (right) process.

stead of AES-256, since it would be computationally less intensive and, since a new AES key is generated for each message, even if it could be broken, it would only be valid for that single message, but when testing the difference in performance in one of the probe's VPS it was not significant. The performance test consisted of creating a new key, encrypting a pre-defined 500 byte string and decrypting it a thousand times. This test was ran 20 times for AES-128 and AES-256, on a Kernel-based Virtual Machine (KVM) VPS running Ubuntu 16.04 with 1 CPU core at 3GHz. The average execution time of the tests with AES-128 was 1.11 seconds and 1.20 seconds for AES-256. AES-128 is only 7.5% faster than AES-256 while offering significantly less security. This performance difference could be major if these probes sent tens of thousands of requests per second, but that is not the case, and so AES-256 was chosen.

Now that both algorithms' usage has been explained let us take a closer look at how they are used in the platform. Whenever a new message needs to be sent, a new AES key is generated and the data is then encrypted using that key. Once that encryption is done, it is then signed using the sender's RSA private key. The AES key is then encrypted using the destination's public key, and the message (Figure 4.6) is constructed and finally sent to the Mainframe. Upon receiving a message, the Mainframe will check for the probe's ID in the message field *ID* and load that probe's public key (if it is not loaded yet). It will then proceed to verify the signature, discarding the message if it cannot validate it, and decrypt the AES key with the sender's RSA public key, and finally decrypt the data using the AES key. This concludes the process of encryption-decryption (Figure 5.12).
The usage of AES combined with RSA allows for both data encryption and sender authentication.

5.7 Graphical User Interface

The aim of this section is to dive further into the provided graphical user interface, which is referred to as simply **Dashboard**. It will attempt to explain the thought process behind the creation and usability features, as well as demonstrate what the final interface looks like. As previously mentioned, the dashboard is the point of entry for the users of the platform. It enables them to control the platform, visualise information regarding the configurations made and make changes to several variables of the platform, such as the probes, targets, modules and its users. The front-end was implemented using Twitter's Bootstrap 3, Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) and Javascript/JQuery, while the back-end was implemented using Python Flask. It enables the following:

- Add, delete, edit probes and users.
- Assign targets to probes and respective data gathering modules.
- Control probes (setup, start, restart, stop).
- Add, delete, edit data gathering, data analysis and countermeasures modules.
- Visualise information regarding all of the points above, plus logs and alarms.

Every single bullet of the list above is necessary for the well functioning of the dashboard and, ultimately, of the platform itself. Before diving further into each of the items, let us take a look at the general layout of the dashboard depicted in Figure 5.13. The dashboard is divided into two main areas: the red area is the navigation sidebar, which allows the users to switch pages by clicking on the buttons; the blue area is the information zone, where all the information and or available actions will be displayed. The remaining areas (yellow and green) are the logout and help pages (yellow) and the home button (green). The layout stays the same throughout the dashboard, with small changes in some pages which main contain tabs in the blue area. Now let us start by taking a closer look at the first bullet.

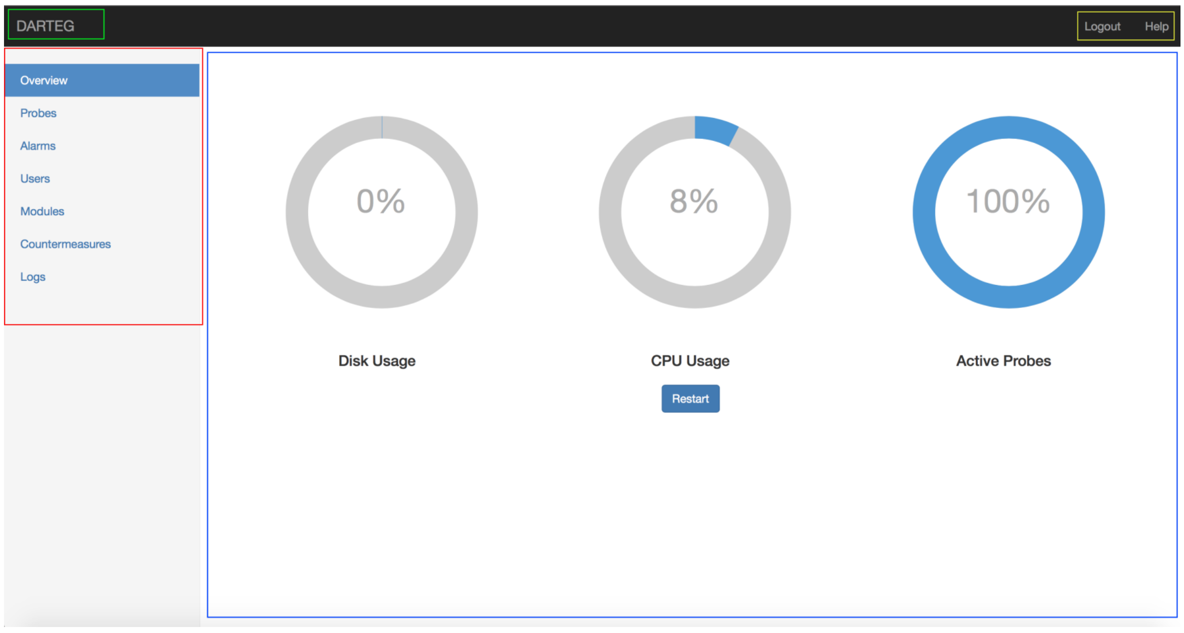
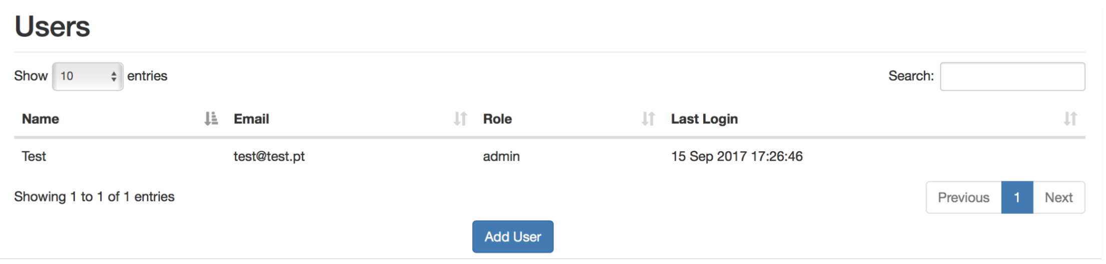


Figure 5.13: Dashboard layout

5.7.1 Users and Probes

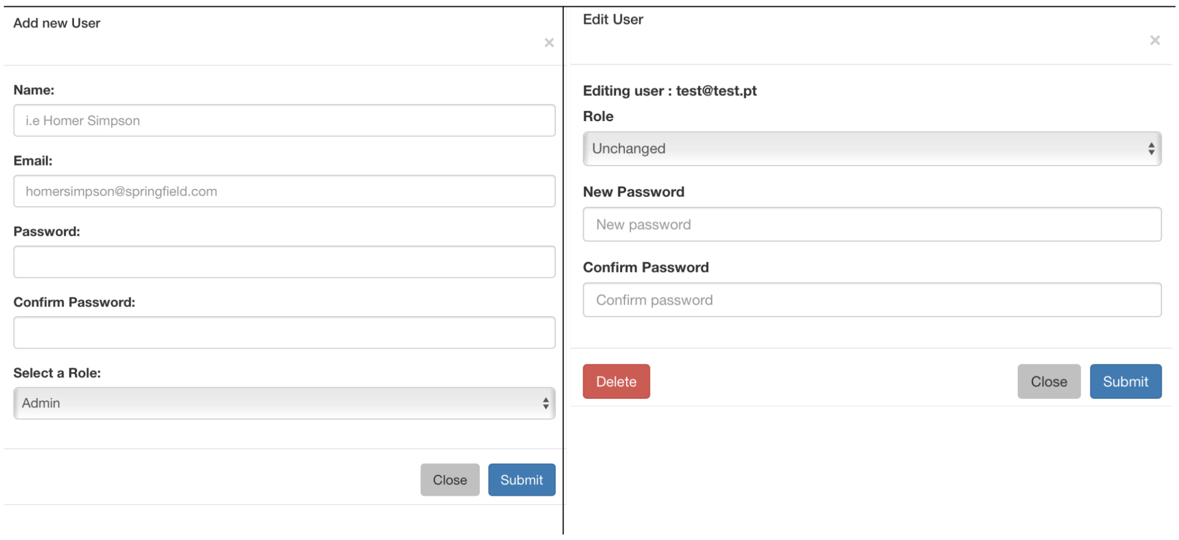
As will be explained in more detail in section **Deployment**, the dashboard will have an admin user that can be created when deploying the Mainframe. With this user, more users can then be added with any level of access (admin, tech or standard) or their details changed (Figure 5.14). Also previously mentioned in chapter **Architecture and Technical Design**, to be able to detect BGP routing anomalies, the platform relies on having a number of probes N spread around the globe, with $N > 1$. That being the case, the first step in the implementation was to create the means to achieve the ability to add probes to the platform. The probes have several fields, some unknown directly to the users, but the ones that must be supplied by the users are: IP address, an alias (to simplify visualisation) and if it is a backup probe for an existing one. Backup probes will be activated if, and only if, their assigned probe goes offline without it being a direct consequence of a user's action. Some extra fields are the city where it is located, country and Linux distribution. All of the former are merely optional and do not affect the usability in any way. This Probes page also provides the ability to add targets, change the modules assigned to each Probe $\rightarrow$ Target pair and execute actions on the probes. The first and only available action after adding a probe is to set it up. For that, one simply needs to click on the desired probe, enter the username and password into the boxes and click the setup icon. After the setup is done, other actions are available. It is then possible to start, stop or restart the probes by, once again, clicking on it. The ability to delete is also present. All of these can be visualised in Figure 5.15. These two pages provide all the pointed out features present on the first three bullets of the list previously presented.


Users

Show 10 entries Search:

Name	Email	Role	Last Login
Test	test@test.pt	admin	15 Sep 2017 17:26:46

Showing 1 to 1 of 1 entries


Add new User

Name: i.e Homer Simpson

Email: homersimpson@springfield.com

Password:

Confirm Password:

Select a Role: Admin

Edit User

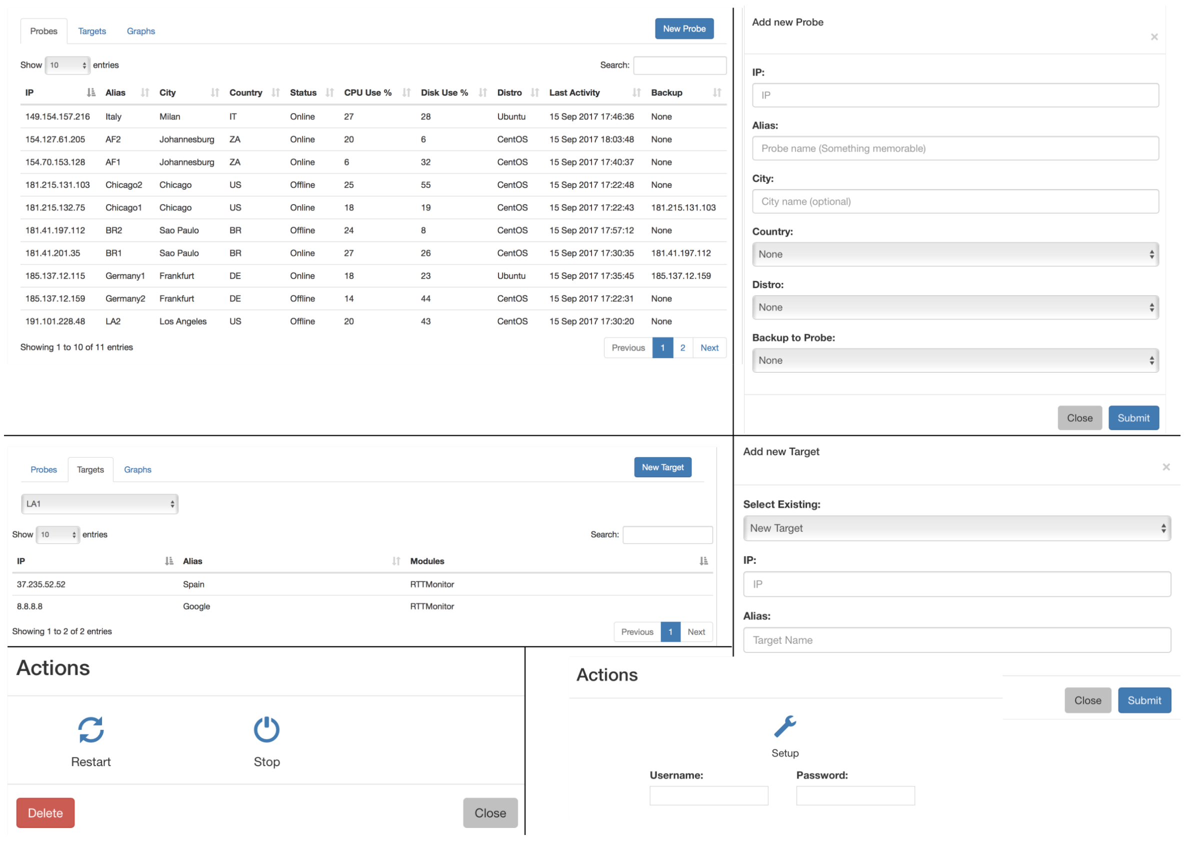
Editing user : test@test.pt

Role: Unchanged

New Password: New password

Confirm Password: Confirm password

Figure 5.14: Users page.



Show 10 entries

Search:

IP	Alias	City	Country	Status	CPU Use %	Disk Use %	Distro	Last Activity	Backup
149.154.157.216	Italy	Milan	IT	Online	27	28	Ubuntu	15 Sep 2017 17:46:36	None
154.127.61.205	AF2	Johannesburg	ZA	Online	20	6	CentOS	15 Sep 2017 18:03:48	None
154.70.153.128	AF1	Johannesburg	ZA	Online	6	32	CentOS	15 Sep 2017 17:40:37	None
181.215.131.103	Chicago2	Chicago	US	Offline	25	55	CentOS	15 Sep 2017 17:22:48	None
181.215.132.75	Chicago1	Chicago	US	Online	18	19	CentOS	15 Sep 2017 17:22:43	181.215.131.103
181.41.197.112	BR2	Sao Paulo	BR	Offline	24	8	CentOS	15 Sep 2017 17:57:12	None
181.41.201.35	BR1	Sao Paulo	BR	Online	27	26	CentOS	15 Sep 2017 17:30:35	181.41.197.112
185.137.12.115	Germany1	Frankfurt	DE	Online	18	23	Ubuntu	15 Sep 2017 17:35:45	185.137.12.159
185.137.12.159	Germany2	Frankfurt	DE	Offline	14	44	CentOS	15 Sep 2017 17:22:31	None
191.101.228.48	LA2	Los Angeles	US	Offline	20	43	CentOS	15 Sep 2017 17:30:20	None

Showing 1 to 10 of 11 entries

Add new Probe

IP: IP

Alias: Probe name (Something memorable)

City: City name (optional)

Country: None

Distro: None

Backup to Probe: None

LA1

Show 10 entries

Search:

IP	Alias	Modules
37.235.52.52	Spain	RTTMonitor
8.8.8.8	Google	RTTMonitor

Showing 1 to 2 of 2 entries

Add new Target

Select Existing: New Target

IP: IP

Alias: Target Name

Username:

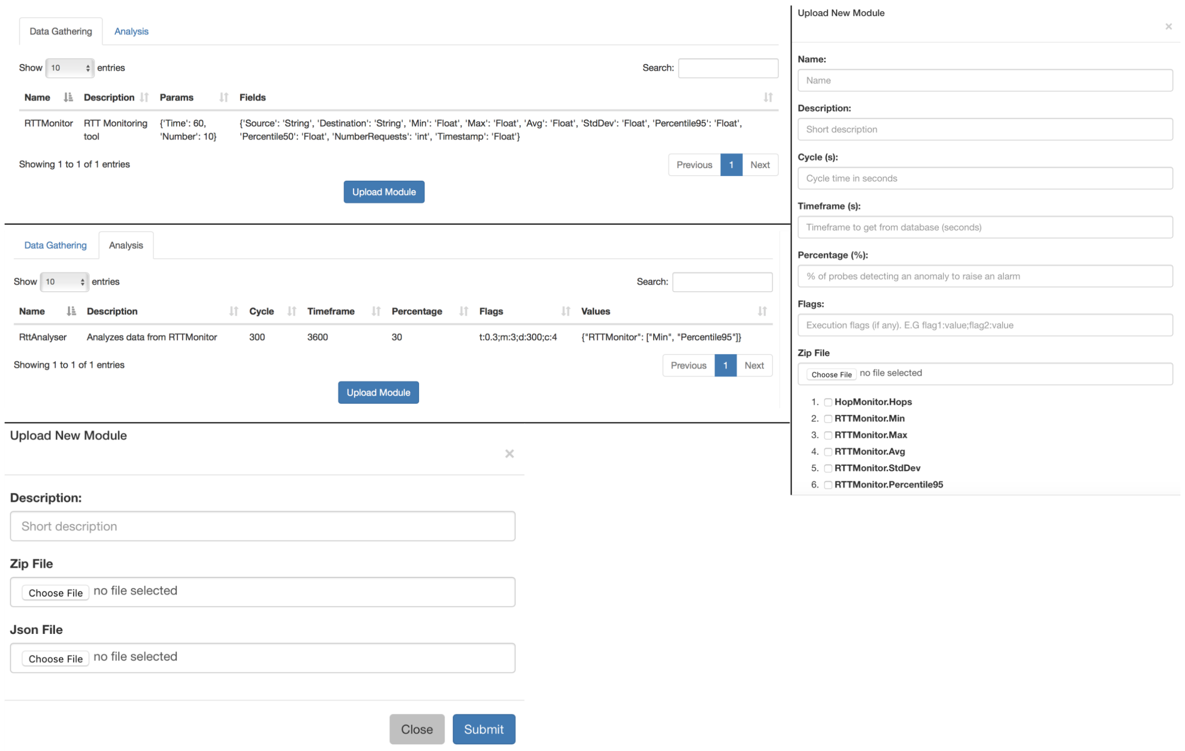
Password:

Figure 5.15: Probes page.

5.7.2 Modules

Another major feature of the platform is the ability to upload, delete and edit modules. In this section, the three types of modules that can be uploaded to the platform will be explained in further detail, which are: data gathering, data analysis and countermeasures. These are the modules responsible for data collection, data analysis and countermeasures to be used when an anomaly is detected. Apart from the countermeasures modules, the other two are essential for this platform to be of any use since without data there is no information that can be extracted and without analysis, raw data is mostly useless. Since it has already been explained what the requirements for uploading each of the modules are, let us simply take a look at how they can be graphically uploaded, edited and removed. Figure 5.16 depicts the module's page that allows operations on data gathering and data analysis modules. On the upper left corner, the first two images list and allow the upload of data gathering and data analysis modules (top and bottom, respectively). It is possible to visualise information regarding the modules, such as the name, description, parameters (execution) and fields of the data gathering modules, as well as the name, description, cycle, timeframe, percentage, flags and values selected for the data analysis modules. To edit or delete the modules, simply click on the desired module, which will present the options seen on Figure 5.17. To the upper right corner is the modal presented to the user when uploading a new data analysis module, it allows the user to enter the name, description, cycle (seconds), timeframe (seconds), percentage (of probes running the module that have to flag an anomaly for it to be considered global) and execution flags (to be passed as argument when executing the module), as well as the fields from data gathering modules that can be selected in order to be received via the TSV file. Finally, in the lower left corner is the modal presented to the user in order to upload a data gathering module.

The countermeasures modules have their own page, which allows the users to upload new countermeasures and assign any number of them to any of the targets. In Figure 5.18 it is possible to see the countermeasures page. On top, information regarding all targets currently in the platform and assigned countermeasures to each one of them can be visualised. To alter the assigned measures, simply click on the desired target and a modal will appear with the options available (bottom left). The image in the middle shows the currently existing countermeasures modules' names and descriptions, and allows the user to upload new ones by clicking on the *New Countermeasure* button. This will show a modal with the fields *name*, *description* and two file upload fields, one for the script to run when an anomaly is detected and another one to revert the changes (optional); these can be seen in the bottom right image.


Data Gathering | Analysis

Show 10 entries Search:

Name	Description	Params	Fields
RTTMonitor	RTT Monitoring tool	{'Time': 60, 'Number': 10}	{'Source': 'String', 'Destination': 'String', 'Min': 'Float', 'Max': 'Float', 'Avg': 'Float', 'StdDev': 'Float', 'Percentile95': 'Float', 'Percentile50': 'Float', 'NumberRequests': 'int', 'Timestamp': 'Float'}

Showing 1 to 1 of 1 entries Previous 1 Next

Upload Module

Data Gathering | Analysis

Show 10 entries Search:

Name	Description	Cycle	Timeframe	Percentage	Flags	Values
RttAnalyser	Analyzes data from RTTMonitor	300	3600	30	t:0.3;m:3;d:300;c:4	{"RTTMonitor": ["Min", "Percentile95"]}

Showing 1 to 1 of 1 entries Previous 1 Next

Upload Module

Upload New Module

Description:
Short description

Zip File
Choose File no file selected

Json File
Choose File no file selected

Close Submit

Upload New Module

Name:
Name

Description:
Short description

Cycle (s):
Cycle time in seconds

Timeframe (s):
Timeframe to get from database (seconds)

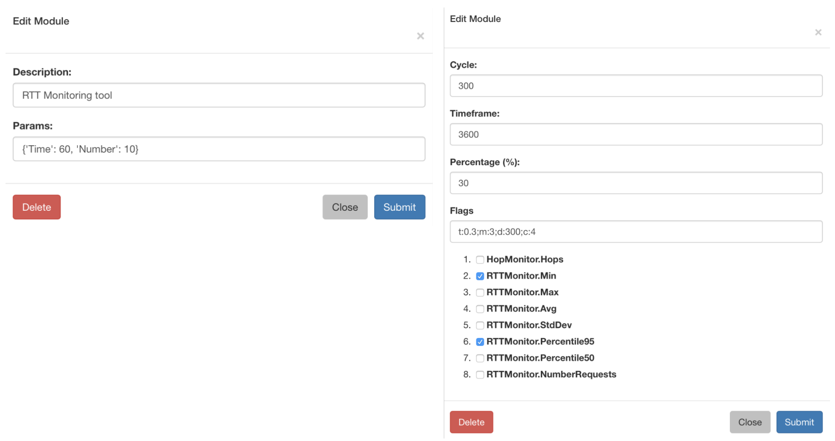
Percentage (%):
% of probes detecting an anomaly to raise an alarm

Flags:
Execution flags (if any). E.G flag1:value;flag2:value

Zip File
Choose File no file selected

1. ☐ HopMonitor.Hops
2. ☐ RTTMonitor.Min
3. ☐ RTTMonitor.Max
4. ☐ RTTMonitor.Avg
5. ☐ RTTMonitor.StdDev
6. ☐ RTTMonitor.Percentile95

Figure 5.16: Data gathering & Data analysis page.


Edit Module

Description:
RTT Monitoring tool

Params:
{'Time': 60, 'Number': 10}

Delete Close Submit

Edit Module

Cycle:
300

Timeframe:
3600

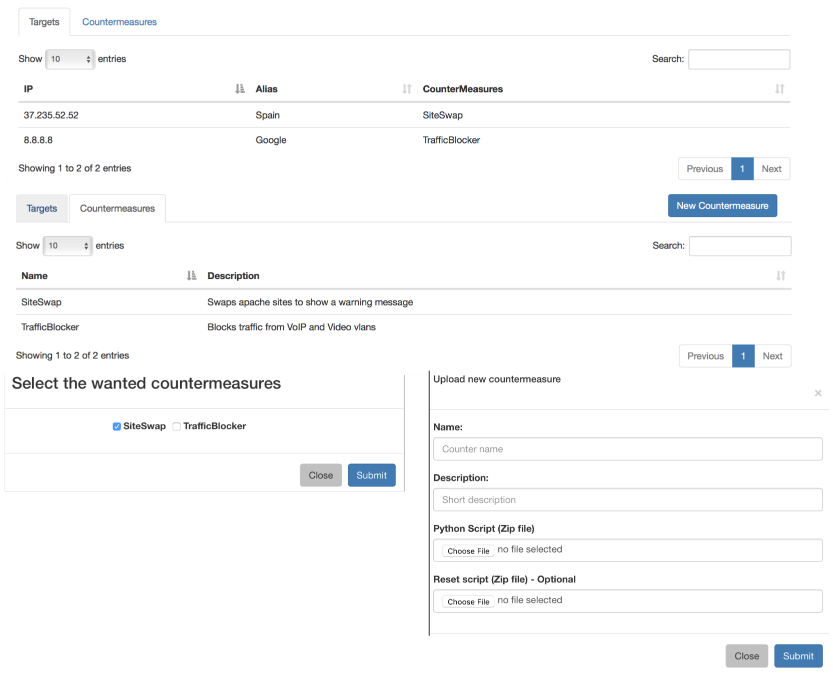
Percentage (%):
30

Flags
t:0.3;m:3;d:300;c:4

1. ☐ HopMonitor.Hops
2. ☑ RTTMonitor.Min
3. ☐ RTTMonitor.Max
4. ☐ RTTMonitor.Avg
5. ☐ RTTMonitor.StdDev
6. ☑ RTTMonitor.Percentile95
7. ☐ RTTMonitor.Percentile50
8. ☐ RTTMonitor.NumberRequests

Delete Close Submit

Figure 5.17: Data gathering (left) & Data analysis (right) edit options.



Show 10 entries Search:

IP	Alias	CounterMeasures
37.235.52.52	Spain	SiteSwap
8.8.8.8	Google	TrafficBlocker

Showing 1 to 2 of 2 entries

Show 10 entries Search:

Name	Description
SiteSwap	Swaps apache sites to show a warning message
TrafficBlocker	Blocks traffic from VoIP and Video vlans

Showing 1 to 2 of 2 entries

Select the wanted countermeasures

[✓ SiteSwap] TrafficBlocker

Upload new countermeasure

Name: Counter name

Description: Short description

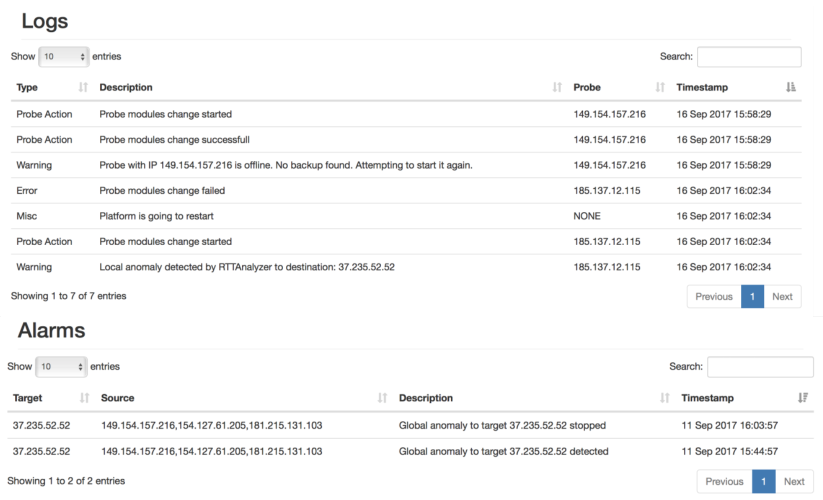
Python Script (Zip file): Choose File no file selected

Reset script (Zip file) - Optional: Choose File no file selected

Figure 5.18: Counter measures page.

5.7.3 Logs and Alarms

While the other pages provide both data visualisation as well as the needed features to manage the platform, these two pages (Logs and Alarms) simply provide information. They were created with the goal to provide extra information regarding what is happening on the platform (logs) as well as giving information regarding detected global anomalies (alarms). The former page provides different types of information: probe related actions, errors, warnings and misc (anything that does not fall into the three previous types). The probe related actions encompass probes setup, start, stop, restart and modules changes; the error type logs any event where an unexpected failure occurred such as, but not limited to, module failure to start and changes that could not be performed on the probes or database; the warning type logs information regarding probes being down, backup probes unable to be started and detection of local anomalies; finally, the last type logs information such as the addition or deletion of probes, platform restarts and countermeasures deployed. These logs provide an extra layer of information for the users, providing a better view of what is happening on the platform. The alarm page provides only one type of information, but it could be argued that it is the most important information, taking into the account the goal of the platform, which is the detection of global anomalies. Whenever a global anomaly is detected, it will be shown on that page, as well as when it is over. Figure 5.19 shows both pages. On the Logs page, there is a table with four distinct fields: *Type*, *Description*, *Probe* and *Timestamp*. The description field provides information on what happened and, if it is related to a probe, the probe field will have that probe's IP; the timestamp field tells the user when that event happened. The Alarms page also contains a table with 4 fields: *Target*, *Source*, *Description* and *Timestamp*. The target field refers to which target the alarm is related to; the source field contains the IPs of all the probes from which the data that led to the detection of the anomaly came from; the last two fields let the users know if it started or stopped and when.



Logs

Show 10 entries Search:

Type	Description	Probe	Timestamp
Probe Action	Probe modules change started	149.154.157.216	16 Sep 2017 15:58:29
Probe Action	Probe modules change successfull	149.154.157.216	16 Sep 2017 15:58:29
Warning	Probe with IP 149.154.157.216 is offline. No backup found. Attempting to start it again.	149.154.157.216	16 Sep 2017 15:58:29
Error	Probe modules change failed	185.137.12.115	16 Sep 2017 16:02:34
Misc	Platform is going to restart	NONE	16 Sep 2017 16:02:34
Probe Action	Probe modules change started	185.137.12.115	16 Sep 2017 16:02:34
Warning	Local anomaly detected by RTTAnalyzer to destination: 37.235.52.52	185.137.12.115	16 Sep 2017 16:02:34

Showing 1 to 7 of 7 entries Previous 1 Next

Alarms

Show 10 entries Search:

Target	Source	Description	Timestamp
37.235.52.52	149.154.157.216,154.127.61.205,181.215.131.103	Global anomaly to target 37.235.52.52 stopped	11 Sep 2017 16:03:57
37.235.52.52	149.154.157.216,154.127.61.205,181.215.131.103	Global anomaly to target 37.235.52.52 detected	11 Sep 2017 15:44:57

Showing 1 to 2 of 2 entries Previous 1 Next

Figure 5.19: Logs (top) and Alarms (bottom) pages.

5.7.4 Back office

The dashboard design, and what it is able to do was shown in the previous sections, but most of it would not be possible without a back office to provide access to actions and information. Any non-static website requires a back office from which to retrieve, insert and modify data, and or perform actions, and, as previously said, most of it has been implemented using Python Flask framework and, although it does provide the means to create APIs to retrieve and insert data (with the help of forms), it alone was not sufficient to create the dashboard. To get the desired features and performance, Javascript/JQuery (from now on mentioned as simply JS) in conjunction with all that Python Flask offers was used. JS utilises the APIs provided by Python-Flask to do two types of requests: POST and GET requests. JS POST requests (Figure 5.20) are used to ask the back office to perform a certain action, such as starting, stopping, restarting or setting up a probe or delete users, probes, targets, etc.; they are mainly used to either launch actions or remove data. The JS GET requests (Figure 5.21) on the other hand are used to ask for data from the back office in order to fill the tables that can be seen in the sections above as well as selectors. The remaining input, such as adding users, probes, targets and modules is done with the use of Flask's forms. These forms contain fields that can be filled by the user and then sent, via POST request, to the respective APIs. These could have also been done with JS POST requests, but since Flask provides the ability to do so without JS, it was decided to take advantage of it. Together, all of these provide the functionality desired for the dashboard.

As previously stated, the platform runs a series of jobs to keep track of various elements and acts upon that information whenever necessary, which takes place in the back office. Since a big part of the platform's objective is to be modular, there are some issues to consider. The ability to upload new modules and execute them is not the most challenging part; however, to be able to upload a data gathering module and create python code that allows Object-relational Mapping (ORM) access to that module's database collection is a bit more challenging. It was already explained how the collection and respective methods are created, but it was yet not explained how the back office actually uses the newly created ORM class.

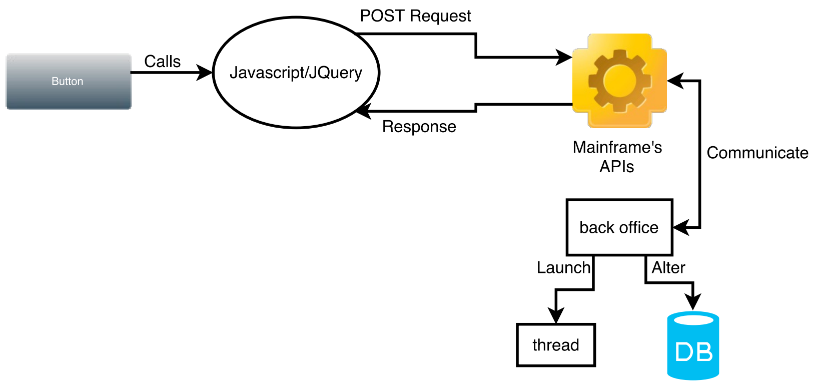


Figure 5.20: Javascript POST request

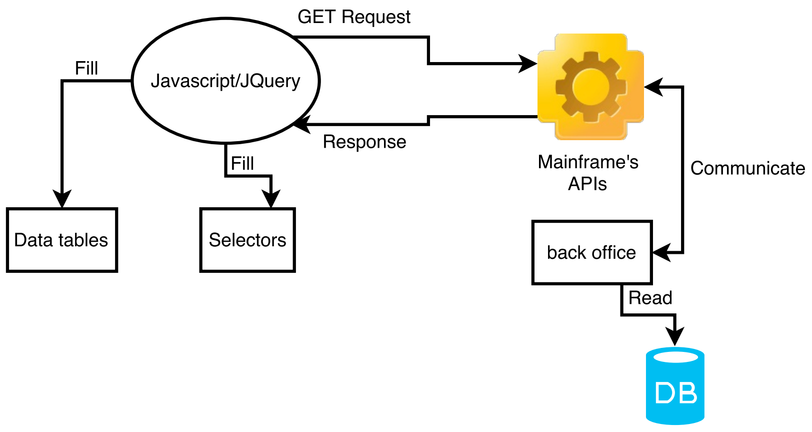


Figure 5.21: Javascript GET request

When inserting or retrieving data using the ORM classes, the name of the class and methods needs to be known in order to be evoked. Since the platform does not know before hand what the names of the modules will be, it has to still be able to evoke and use their class as well as their methods. To achieve that, the built-in Python's method *eval()* was used. *eval()* accepts a string as argument and is able to execute it (if it is a method or function). By using that, together with pre-defined methods signatures and class names (such as Mod_*ModuleName* and mod_addTo*RTTMonitor*), the platform is able to manipulate the database without knowing before hand which module made the request to the APIs. To better illustrate this, let us take a look at the code snippet below:

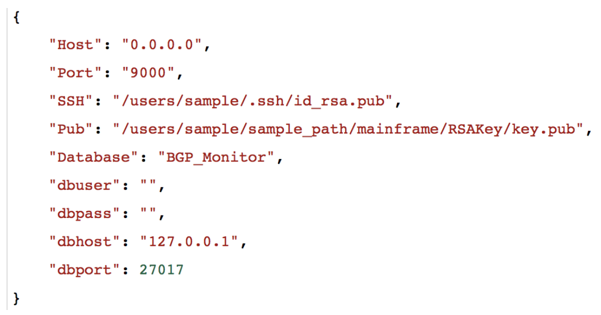
```
def receiveData():
        """
        API to receive data from probes
        """
        data = request.get_json()
        pub_key = loadClientRSAPub(data["ID"])
        try:

            probeID = data["ID"]
            signature = base64.b64decode(data["Signature"].encode("utf-8"))
            aes_key       = base64.b64decode(data["AESKey"].encode("utf-8"))
            tag       = base64.b64decode(data["Tag"].encode("utf-8"))
            nonce       = base64.b64decode(data["Nonce"].encode("utf-8"))
            encrypted_data = base64.b64decode(data["Data"].encode("utf-8"))
            data = json.loads(crypto_handler.decryptAndVerify(pub_key,
    private_key,signature,aes_key,encrypted_data,tag,nonce).decode('UTF-8'))
            probe = Probes.objects(ID=probeID).first()
            data["Source"] = probe.IP
            probe.LastActivity = time.time()
            probe.Status = 1
            probe.save()
            eval("mod_addTo"+data["ModuleName"]+"(data)")
        except Exception as e:
            return json.dumps({"Success":0}),403
        return json.dumps({"Success":1}),200
```

This snipped shows the API that receives data from the Probes. As can be seen, in order to add a new document of the module's collection to the database, it makes use of the methods automatically created when the module was uploaded, by evoking the eval() function. The same can be done to retrieve data by, once again, making use of the eval() function. This solves the issue of data insertion and retrieval when one does not know in advance the names of the classes and methods.

5.8 Deployment

The aim of this section is to elaborate on how to deploy the platform. Since the Mainframe has the ability to take the necessary actions to setup probes, all that is required to do is to install the Mainframe. Since it requires Python 3+ and MongoDB, the server unto which the users intend to install the platform must have those installed beforehand. The platform can be installed to run as a standalone, e.g., without any **Web Server Gateway Interface (WSGI)** although it is not recommended to do so since Flask itself is not meant to be deployed without the use of a WSGI [44]. Since there are several options regarding WSGIs, it is up to the user to choose and deploy one and, as such, no automation is provided in this case (regarding the WSGI itself). To install the platform, a python script is provided together with a .zip file, which contains all necessary code and folder structure required to run the Mainframe. The script will start by extracting the contents of the .zip file to the current directory from which the script was executed and proceed to create a python virtual environment within the root folder of the Mainframe. The script will then install all the requirements present in the *requirements.txt* file into the virtual environment and create an RSA-4096bits key pair on a sub-folder called RSAKey; thus concluding the installation process. With this out of the way, now it is time to edit the *flask_configs.txt* and *mainframe.json* files. The former contains all of the needed information for the platform to run, which is: Host, Port, SSH, Pub, Database, dbuser, dbpass, dbhost and dbport. The Host and Port tells Flask on what IP and port it should start listening for requests; the SSH field points to the SSH public key to be written in the probes authorized_keys file (to be able to login without password); Pub points to the RSA public key to be transferred to the probes (which is located in a folder named RSAKey in the root path of the Mainframe together with the private key); The Database field is used to tell Flask which database to use; the remaining fields, dbuser, dbpass, dbhost and dbport are the user, password, host and port of the MongoDB to which



```
{
    "Host": "0.0.0.0",
    "Port": "9000",
    "SSH": "/users/sample/.ssh/id_rsa.pub",
    "Pub": "/users/sample/sample_path/mainframe/RSAKey/key.pub",
    "Database": "BGP_Monitor",
    "dbuser": "",
    "dbpass": "",
    "dbhost": "127.0.0.1",
    "dbport": 27017
}
```

Figure 5.22: Sample flask_configs.txt file

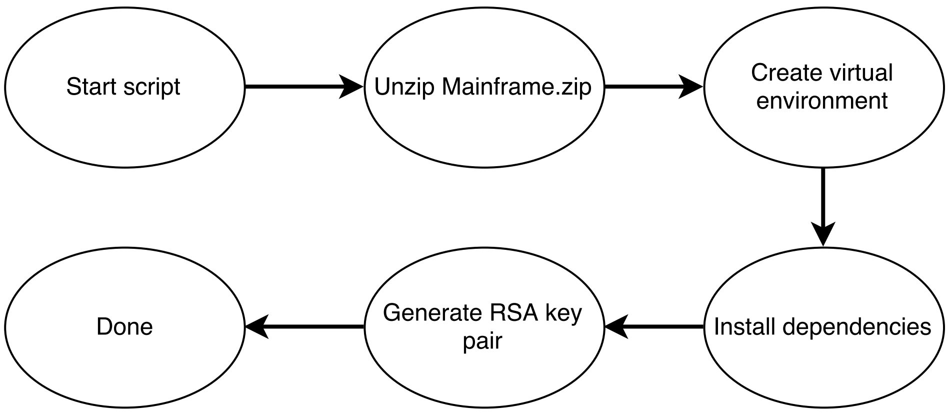


Figure 5.23: Setup script flow diagram.

the Mainframe will connect. A sample configuration file can be seen depicted in Figure 5.22. The second file, *mainframe.json*, contains three fields: IP, Port and SSL. These fields are the IP address and port from which the Probes can access the Mainframe via HTTPS (or HTTP by changing the *SSL* value in *mainframe.json* from 1 to 0). As previously mentioned, this file will be sent to the Probes upon setup. These two files must be altered to the correct values before attempting to deploy the Mainframe since, without them, the platform will not be able to function properly. After this process is complete, the Mainframe can then be started by simply running it by either executing it via the terminal (*./mainframe.py*) or calling it with python (*python mainframe.py*). Now it is up and running, and it can be accessed with a web browser on the URL *ip:host/adminControl*. After the setup is complete, the user of the platform must create a new user to be able to login. The user can be created via a provided piece of code located at *user_scripts*, on the root of the mainframe's folder. All users created via this code have full privileges on the platform. The process of extracting the files, generating keys and setting up the virtual environment can be seen depicted in Figure 5.23.

5.8.1 Requirements

In order to be able to run the platform, there are a few requirements that have to be met. Since the Mainframe is responsible for receiving data from multiple sources, store it, run periodic checks on the platform, analyse data and deploy a dashboard it has heavier hardware requirements than the probes. The minimum requirements advised for running the Mainframe are Quadcore 2.4GHz+, 500GB of disk space, 8GB RAM+, Linux (Ubuntu 16.04+, CentOS6+, Debian7+) as well as Python 3.X, PIP and MongoDB. For the probes, the minimum requirements are Python 3.X, PIP, 128MB (of usable) RAM, single core 1.0GHz+, 10GB of disk space and Linux (Ubuntu 16.04+, Debian 8+, CentOS7+). If the probes are running on a VPS, it must be able to run docker. KVM and Open Virtuozzo (OpenVZ) 7, for example, are able to run docker.

Chapter 6

Results

This chapter will present the results obtained using the platform, how the data was collected and used to validate the methodologies proposed in this dissertation. The results for local, and global detections will be discussed for both methodologies. Platform performance test results will also be presented.

6.1 Anomaly generation

In order to test and validate the implemented analysis module and its respective algorithm it is required to have data. Since the data obtained in the platform may not have any anomalies or if it does, there is no way to validate it without a study case of data, a method that would allow for the generation of that data was required. One of the ways would be to change the routing tables by modifying BGP adverts, but since data is being collected in the real world, such is not possible for technical reasons (no access to ISPs network equipment) and because that would also be disruptive. A BGP routing attack works by effectively changing the path that traffic takes from a source to a destination, so, that was replicated in this dissertation. The path it should take to simulate a BGP routing attack would be from source to relay, relay to destination, and destination back to source. For this to happen, the source IP of the original packet cannot be changed; it must remain the same, but that creates other issues. If the packet's source address at the exit of the network does not match any prefixes of that network, it may be discarded; if it is changed in the relay to the relay's IP address, then the destination will simply answer back to the relay, which is not what is wanted. So, a solution that would keep the original IP headers intact, while providing the ability to do what was needed, was also required. That solution was to use tunnels, more precisely IPv4 Generic Routing Encapsulation (GRE) tunnels ([45]). Figure 6.1 depicts the encapsulation of the packet that takes place when it enters the tunnel. The original headers and data remain unaltered, and a new header is added on top of them.

So, traffic from source to destination is redirected in the source, by applying iptables rules, to the relay which will then also redirect it to the destination via the tunnel, as depicted in red in Figure 6.2 (green represents the normal flow). In this scenario, let us assign the letters S, D and R to source, destination and relay, respectively. When S sends a packet to D, the iptables will catch that packet when it is outbound, and redirect it to R instead. R will then receive

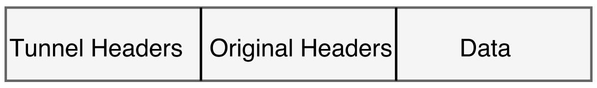


Figure 6.1: IPv4 GRE tunnel encapsulation.

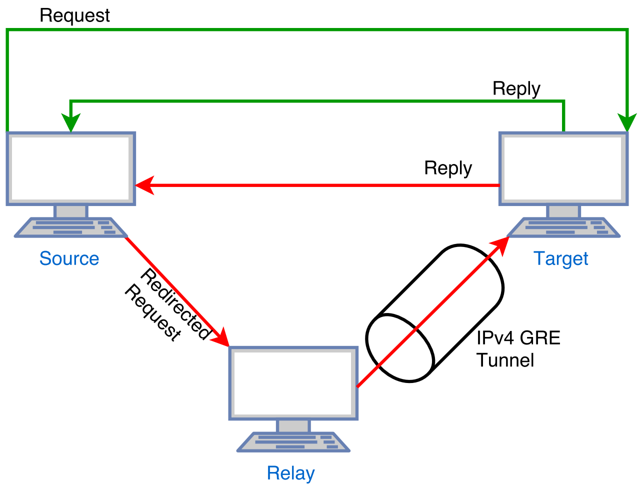


Figure 6.2: Traffic being redirect through an IPv4 GRE tunnel.

---

the packet, and redirect it to the tunnel that has been established with D, and, when entering the tunnel, the original packet will now be encapsulated, and a new header added. Since the original packet had a header with source IP S and destination IP D, the encapsulated packet will keep this header but add a new one on top with source TR and destination TD (where TR is the tunnel's interface IP address on the relay and TD the tunnel's interface IP address on the destination), sending it to the destination via the tunnel. When the destination receives the packet on the tunnel interface, it will extract the tunnel's header, and read the original header (source S and destination D), sending then a response to source S, thus completing the process. To have a clearer view of the process, it is depicted in Figure 6.3. Now that it is known what is needed to simulate an anomaly, these changes have to be implemented on the wanted VPSes.

As previously mentioned, there are several VPSes spread around the globe, 19 to be more precise. These were then split into three groups in order to gather this data: 11 sources, 4 targets and 4 relays. To be able to compare the values obtained with the generated anomaly, there is also a need to gather data without any anomalies, and, as such, the total number of possible combinations is $11 * 4 * 5$ (220). Doing the needed configurations for the 220 possible combinations manually would take a long time, and be unproductive, so several scripts to make the configuration faster, and easier, were created. For more accurate data comparison, the data obtained from a source to a destination with the 4 relays, and with no relay, must be taken at the same time. In order to do so, the iptables were configured to redirect traffic not only by destination IP but also destination port. With that being the case, the gathering module RTTMonitor is not usable since it does not support the use of ports, so another piece of code that was able to do so was created.

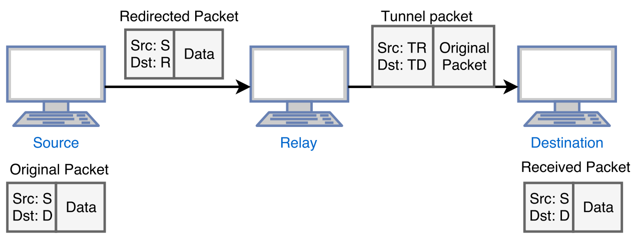


Figure 6.3: Tunnel packet transformation.

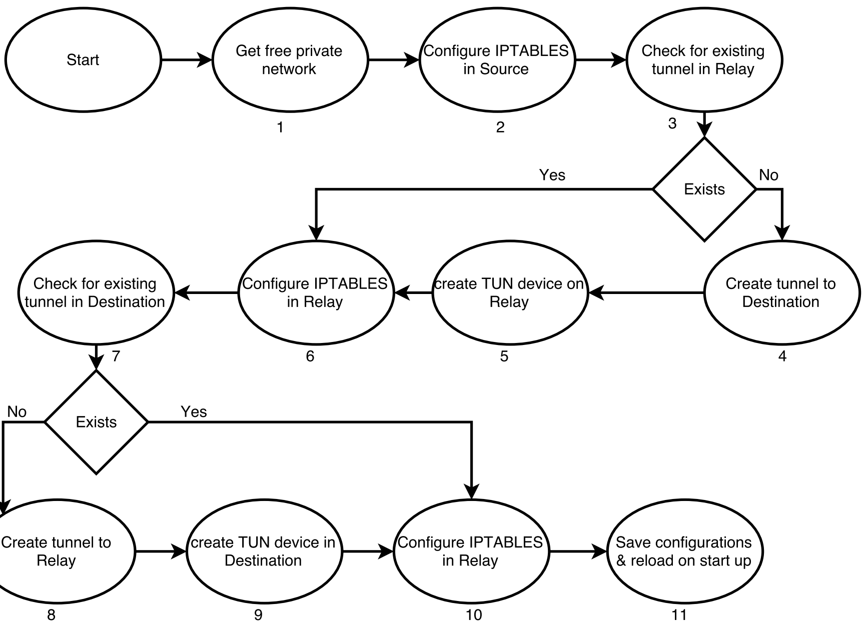

Start
Get free private network
1
Configure IPTABLES in Source
2
Check for existing tunnel in Relay
3
Exists
Yes
No
Create tunnel to Destination
4
create TUN device on Relay
5
Configure IPTABLES in Relay
6
Check for existing tunnel in Destination
7
Exists
No
Yes
Create tunnel to Relay
8
create TUN device in Destination
9
Configure IPTABLES in Relay
10
Save configurations & reload on start up
11

Figure 6.4: Configuration diagram to generate anomalies.

Firstly, it is required to explain the configuration scripts for a better understanding of how these configurations were made, with the help of a diagram (Figure 6.4). In order to create a tunnel between two networks, or terminals, a subnet that is not in use in either networks or terminals is required, so that it can be used by the tunnel, and that is what the script does on step 1: retrieves the networking information from the interfaces of Relay and Destination via SSH, and finds a subnet that is free on both terminals. If step 1 fails, the script cannot go on and terminates but, otherwise, it will start by configuring the Source's iptables OUTPUT chain to redirect packets by IP:Port to the appropriate Relay. After step 2 is done, it will then check for an existing tunnel from Relay to Destination (step 3), and, if it does not exist, create the tunnel (step 4) and a TUN interface (step 5), configuring that TUN interface with the IP given by the script from the free subnet, and adding a new route to the Destination's IP via the tunnel; if it exists, it goes to step 6, and configure the Relay's iptables PREROUTING chain to redirect packets to the tunnel based on the port of the incoming packet. The same is then done for the Destination as it was for the Relay regarding the tunnel and TUN, without any changes to the iptables. The configurations are saved and added to sysctl to be reloaded if the VPSes go down. All of this is done via SSH, and with the use of Linux tools only.

As explained previously, RTTMonitor is not capable of doing what is required to collect data with anomalies, since to do so it would need to be able to send requests with a TCP port number as well, and ICMP requests are layer 3, not layer 4. So, with the usage of a Linux tool called *hping3*, which is able to provide similar results as *ping* but with the usage of TCP and ports, a script to capture the data required was developed. It is named *Hping3Main*, and it receives a number of arguments: *-t --time* interval (seconds) to repeat the requests, *-d --dst* list of destinations with ports ([IP:PORT]), *-n --number* number of requests to send to each destination per loop, *-f --fileNames* file names to save the results to. The output of this script is the minimum, maximum, average, standard deviation, median and percentiles 50 and 95, all of these are RTT values. It also outputs a time stamp. The reasons this script was not used as a data gathering module is that, firstly, it requires to be ran as root, and, secondly, in a real-world environment most TCP ports are closed from the outside, unlike ICMP, so it would require that the Targets would open these ports to be able to collect data, which could be a hassle or, more importantly, a security risk. This script is then able to collect the data that is required, and it was deployed to all Sources.

To validate the second methodology (3.2.2), the data was obtained in two different ways. For the first part of the methodology, the part which compares the path, all the sources and relays were configured to run traceroutes to all the destinations. The sources were also configured to run traceroutes to the relays. The output of the script used for the capture of data is the time stamp, a vector containing the different **ASNs** through which traffic flows (in order from source to destination), and the RTT values for each of the hops (of different **ASNs**) in the path taken. These were then assembled to create the respective path from source to destination with the several anomalies. It had to be done this way due to some routers along some paths not responding back to the traceroute when using TCP packets. As such, it was not possible to configure the sources to perform multiple traceroutes to all the destinations with different ports since TCP could not be used. For the second part of the methodology, which is only performed when the path being analysed is a match for previously known paths, a different way of capturing data is required. This part of the methodology attempts to detect the perfect case scenario attack, where the attacker controls a router belonging to the normal path of traffic flow from one, or several, networks to the victim. That scenario was previously mentioned in 3.2.2, and can be seen in Figure 3.4. To simulate this, a tunnel was created from a probe to another probe, simulating the anomaly along the path, without it affect the path. This can be seen in Figure 6.5. The best scenario would have been to create a network of tunnels with all the probes, simulating the same behaviour as in Figure 3.4. Unfortunately, that was not possible to do on time.

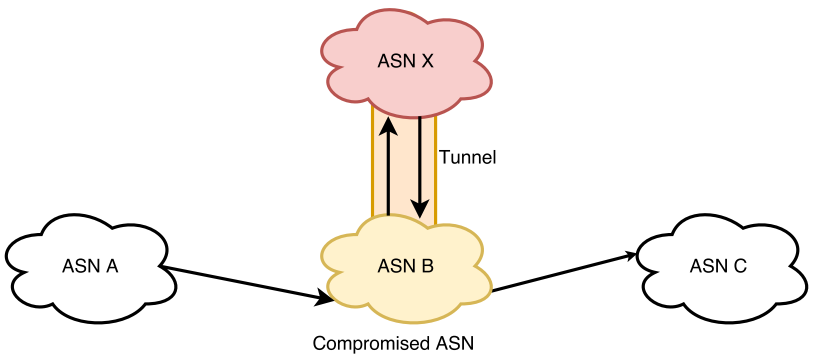


Figure 6.5: Use of a tunnel to simulate a perfect BGP attack.

6.2 Methodology 1 results

Probe List			
Source	Network	Origin AS	Provider
Amsterdam (Netherlands, EU)	151.236.28.0/24	AS43350	Host1Plus (OVZ)
Chicago 2 (Chicago, IL, USA)	181.215.128.0/19	AS61440	Host1Plus (OVZ)
Chile (Chile,SA)	37.235.52.0/24	AS28099	EDIS (KVM)
Frankfurt 2 (Germany, EU)	185.137.12.0/22	AS61317	Host1Plus (OVZ)
Iceland (Iceland, EU)	37.235.49.0/24	AS50613	EDIS (KVM)
Israel (Israel, Middle East)	193.182.144.0/24	AS61102	EDIS (KVM)
LA 2 (Los Angeles, CA, USA)	191.101.224.0/20	AS61440	Host1Plus (OVZ)
Milan (Milan, Italy, EU)	149.154.157.0/24	AS20836	EDIS (KVM)
South Africa 1(Johannesburg, South Africa)	154.70.152.0/22	AS37692	Host1Plus (OVZ)
South Africa 2 (Johannesburg, South Africa)	154.127.61.0/24	AS37692	Host1Plus (OVZ)
Sweden (Sweden, EU)	178.73.192.0/18	AS42708	EDIS (KVM)
Targets	Network	Origin AS	Provider
Chicago 1 (Chicago, IL, USA)	181.215.128.0/19	AS61440	Host1Plus (OVZ)
Frankfurt 1 (Germany, EU)	185.137.12.0/22	AS61317	Host1Plus (OVZ)
Hong Kong (China)	158.255.208.0/24	AS57169	EDIS (KVM)
London (UK, EU)	46.17.56.0/21	AS39326	EDIS (KVM)
Relays	Network	Origin AS	Provider
LA 1 (Los Angeles, CA, USA)	191.101.224.0/20	AS61440	Host1Plus (OVZ)
Madrid (Spain, EU)	37.235.53.0/24	AS39020	EDIS (KVM)
Moscow (Russia)	213.183.56.0/24	AS56630	EDIS (KVM)
SP1 (São Paulo, Brazil, SA)	181.41.201.0/24	AS61440	Host1Plus (OVZ)

Table 6.1: Source, Target and Relay locations, IP network, autonomous system, provider and virtualization technology.

As previously explained, this methodology relies on a set of probes and target networks and uses the RTT values obtained by the probes to detect anomalies. The data was collected from 21st of May, 2017 to 17th of June, 2017 with the use of 19 probes spread over the globe. This section will present part of the collected data, results, and conclusions about the capability of the platform to detect these anomalies by utilising the already mentioned data gathering and data analysis modules regarding the first methodology (3.2.1). Table 6.1 shows the locations, IP networks, autonomous systems, provider and virtualisation technology of the probes, as well as their division in 3 groups: Sources, Targets and Relays. When analysing the data to detect BGP routing anomalies, the aim is to detect the most number of anomalies while trying to minimise false alarms. In order to do so, the data was tested with several different values for *threshold* (in this section every reference to local threshold will simply be *threshold* unless stated otherwise), *maxOccur* and *timeframe*. The *threshold* is the value used to set the line from which an RTT measurement is anomalous or not (above or below the threshold, respectively), *maxOccur* is the number of consecutive occurrences before it is considered an anomaly (or reverted to normal), and the *timeframe* is the time window of collected data that can be used to calculate the average value (to be used as a base value for the comparison). The data was analysed starting with 8 days of 'normal' routing, with intervals of 2 days with routing anomalies, followed by a day of normal routing for the 4 anomalies. Let us start by looking how these values affect the local detection and then move on to the global detection of anomalies.

6.2.1 Local detection

Threshold	100%			60%			30%			25%		
Source	D	U	F	D	U	F	D	U	F	D	U	F
Iceland	51.66	48.34	0.0	74.98	25.02	0.0	77.51	22.49	0.0	87.05	12.95	0.0
Amsterdam	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Ger2	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Sweden	75.36	24.64	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Milan	88.4	11.6	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Chicago2	25.25	74.75	0.0	25.25	74.75	0.0	75.09	24.91	0.0	75.09	24.91	0.0
LA2	0.32	99.68	0.0	25.24	74.76	0.0	25.24	74.76	0.0	48.85	51.15	0.0
Chile	0.95	99.05	0.8	2.18	97.82	1.4	3.6	96.4	1.8	8.3	91.7	1.84
SA 1	25.24	74.76	0.0	50.16	49.84	0.0	50.16	49.84	0.0	51.79	48.21	0.0
SA 2	25.24	74.76	0.0	50.16	49.84	0.0	51.79	48.21	0.0	52.55	47.45	0.0
Israel	50.16	49.84	0.0	77.43	22.57	0.0	100.0	0.0	0.0	100.0	0.0	0.0

Threshold	20%			15%			10%			5%		
Source	D	U	F	D	U	F	D	U	F	D	U	F
Iceland	100.0	0.0	0.0	100.0	0.0	0.09	100.0	0.0	0.66	100.0	0.0	0.66
Amsterdam	100.0	0.0	0.0	100.0	0.0	0.16	100.0	0.0	1.37	100.0	0.0	32.63
Ger2	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Sweden	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Milan	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Chicago2	77.52	22.48	0.0	77.52	22.48	0.0	100.0	0.0	0.0	100.0	0.0	0.0
LA2	52.63	47.37	0.0	75.08	24.92	0.0	75.08	24.92	0.0	76.97	23.03	0.66
Chile	27.98	72.02	1.89	54.2	45.8	1.89	78.61	21.39	1.93	86.59	13.41	1.93
SA 1	56.66	43.34	0.0	77.51	22.49	0.0	100.0	0.0	0.0	100.0	0.0	0.0
SA 2	75.08	24.92	0.0	77.51	22.49	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Israel	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.43	100.0	0.0	0.45

Table 6.2: Local results Germany 1 Target - Timeframe = 7 days, MaxOccur = 5

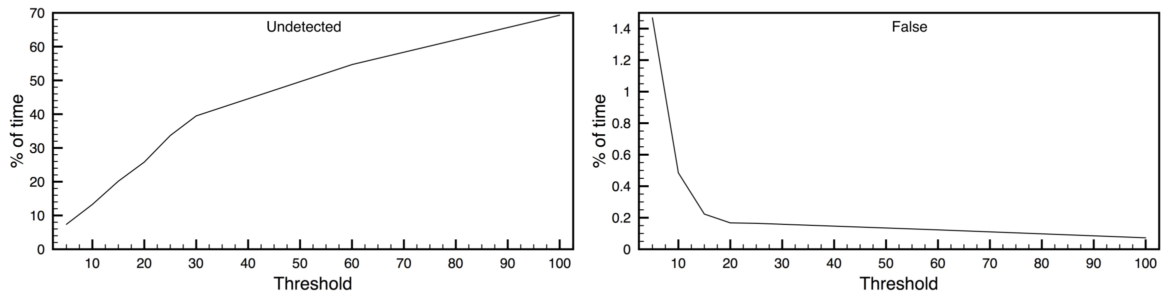


Figure 6.6: Correlation between undetected anomalies (left) and false anomalies (right).

Threshold	100%			60%			30%			25%		
Source	D	U	F	D	U	F	D	U	F	D	U	F
Iceland	0.33	99.67	0.0	0.33	99.67	0.0	25.25	74.75	0.0	25.25	74.75	0.0
Amsterdam	0.33	99.67	0.0	0.33	99.67	0.0	25.25	74.75	0.0	25.25	74.75	0.0
Ger2	0.33	99.67	0.0	0.33	99.67	0.0	25.25	74.75	0.0	25.25	74.75	0.0
Sweden	0.33	99.67	0.0	0.33	99.67	0.0	25.25	74.75	0.0	25.25	74.75	0.0
Milan	0.33	99.67	0.0	0.33	99.67	0.0	25.25	74.75	0.0	25.25	74.75	0.0
Chicago2	0.33	99.67	0.0	75.08	24.92	0.0	75.08	24.92	0.0	75.08	24.92	0.0
LA2	50.42	49.58	0.0	75.08	24.92	0.0	75.08	24.92	0.0	75.08	24.92	0.0
Chile	0.95	99.05	0.0	1.48	98.52	1.11	10.26	89.74	1.6	44.69	55.31	1.63
SA 1	0.37	99.63	0.0	0.37	99.63	0.0	4.62	95.38	0.0	25.25	74.75	0.0
SA 2	0.36	99.64	0.0	0.36	99.64	0.0	8.96	91.04	0.0	25.25	74.75	0.0
Israel	0.33	99.67	0.0	9.01	90.99	0.0	25.25	74.75	0.0	39.92	60.08	0.0
Threshold	20%			15%			10%			5%		
Source	D	U	F	D	U	F	D	U	F	D	U	F
Iceland	25.25	74.75	0.0	25.25	74.75	0.0	52.06	47.94	0.0	84.84	15.16	0.71
Amsterdam	25.25	74.75	0.0	25.25	74.75	0.0	25.25	74.75	0.0	46.38	53.62	0.71
Ger2	25.25	74.75	0.0	25.25	74.75	0.0	27.68	72.32	0.0	50.16	49.84	0.71
Sweden	50.16	49.84	0.0	67.65	32.35	0.28	75.31	24.69	3.16	100.0	0.0	4.91
Milan	61.0	39.0	0.0	100.0	0.0	0.31	100.0	0.0	3.45	100.0	0.0	4.81
Chicago2	75.08	24.92	0.0	75.08	24.92	0.64	92.53	7.47	0.69	100.0	0.0	0.54
LA2	75.08	24.92	0.0	75.08	24.92	0.69	75.08	24.92	0.69	75.08	24.92	0.75
Chile	51.6	48.4	1.67	51.64	48.36	1.84	52.13	47.87	1.87	74.1	25.9	2.12
SA 1	25.25	74.75	0.0	25.25	74.75	0.0	25.28	74.72	0.0	56.8	43.2	0.69
SA 2	25.25	74.75	0.0	25.25	74.75	0.0	25.25	74.75	0.0	74.58	25.42	0.23
Israel	84.29	15.71	0.0	100.0	0.0	0.0	100.0	0.0	2.32	100.0	0.0	3.51

Table 6.3: Local results Hong Kong Target - Timeframe = 7 days, MaxOccur = 5

The tables 6.2, 6.3, 6.4, 6.5, 6.6, 6.7 and 6.8 present data relative to local anomalies detected (**D**), undetected (**U**) and false detections (**F**) in percentage of time relative to the anomaly intervals (**D** and **U**) and the total interval of time (**F**). When looking at the data displayed in the tables, there are a few behaviours that can be detected for the generality of the probes such as that the higher the *threshold*, the higher the undetected anomalies value and the lower the value of false detections. The lower the *threshold*, the higher the value of false detections and the lower the value of undetected anomalies (Figure 6.6).

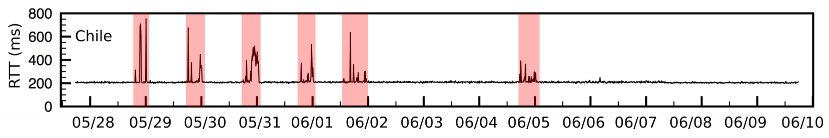


Figure 6.7: Abnormal spikes in RTT caused by hardware or network instability.

Threshold	100%			60%			30%			25%		
Source	D	U	F	D	U	F	D	U	F	D	U	F
Iceland	75.09	24.91	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Amsterdam	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Ger2	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Sweden	50.17	49.83	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Milan	75.08	24.92	0.0	91.3	8.7	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Chicago2	25.25	74.75	0.0	75.08	24.92	0.0	89.89	10.11	0.0	100.0	0.0	0.0
LA2	0.33	99.67	0.0	25.25	74.75	0.0	50.17	49.83	0.0	58.28	41.72	0.0
Chile	1.32	98.68	1.04	2.17	97.83	1.39	5.78	94.22	1.7	28.04	71.96	1.87
SA 1	25.25	74.75	0.0	50.16	49.84	0.0	75.09	24.91	0.0	75.09	24.91	0.0
SA 2	20.0	80.0	0.0	50.16	49.84	0.0	51.79	48.21	0.0	74.84	25.16	0.0
Israel	50.13	49.87	0.0	75.09	24.91	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Threshold	20%			15%			10%			5%		
Source	D	U	F	D	U	F	D	U	F	D	U	F
Iceland	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.16
Amsterdam	100.0	0.0	0.0	100.0	0.0	0.1	100.0	0.0	0.24	100.0	0.0	3.38
Ger2	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.66	100.0	0.0	0.76
Sweden	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Milan	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.31
Chicago2	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
LA2	75.08	24.92	0.0	75.08	24.92	0.0	75.08	24.92	0.0	75.08	24.92	0.0
Chile	41.22	58.78	1.91	53.05	46.95	1.91	90.61	9.39	1.91	100.0	0.0	1.96
SA 1	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
SA 2	75.09	24.91	0.0	75.09	24.91	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Israel	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0

Table 6.4: Local results London Target - Timeframe = 7 days, MaxOccur = 5

There are, however, some instances where that does not show clearly, such as Amsterdam's probe when monitoring London's target. Since the RTT from that probe to the target is, under normal conditions, very low, all the anomalies are detected even with a threshold of 100%, but the trend of a larger value of false detections is still visible (3.38% with a threshold of 5% Table 6.4). The Chicago 1 probe also detects an anomaly with a *threshold* of 100% when it is deviated by São Paulo 1, most likely due to the lower network speeds from Brazil to Europe, but all the others go unnoticed. Some of the more unstable probes, such as Chile, display an abnormal behaviour, raising false detections even with a threshold of 100% to all targets except Hong Kong. This is caused by instability in the probe, may it be hardware or network related. Since the probe is not stable, there are spikes in the RTT values which can induce the probe into thinking there is an anomaly occurring when in reality it is not (Figure 6.7).

Threshold	100%			60%			30%			25%		
Source	D	U	F	D	U	F	D	U	F	D	U	F
Iceland	75.87	24.13	0.0	93.44	6.56	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Amsterdam	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Ger2	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Sweden	51.73	48.27	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Milan	75.87	24.13	0.0	77.43	22.57	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Chicago2	27.59	72.41	0.0	55.41	44.59	0.0	91.2	8.8	0.0	100.0	0.0	0.0
LA2	3.46	96.54	0.0	27.59	72.41	0.0	51.73	48.27	0.0	57.25	42.75	0.0
Chile	3.47	96.53	0.0	3.47	96.53	1.39	3.84	96.16	1.7	18.49	81.51	0.0
SA 1	27.59	72.41	0.0	51.73	48.27	0.0	75.87	24.13	0.0	75.87	24.13	0.0
SA 2	23.53	76.47	0.0	51.73	48.27	0.0	52.57	47.43	0.0	75.87	24.13	0.0
Israel	51.95	48.05	0.0	75.87	24.13	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Threshold	20%			15%			10%			5%		
Source	D	U	F	D	U	F	D	U	F	D	U	F
Iceland	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Amsterdam	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Ger2	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Sweden	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Milan	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Chicago2	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
LA2	75.86	24.14	0.0	75.86	24.14	0.0	75.86	24.14	0.0	75.86	24.14	0.0
Chile	45.04	54.96	0.0	51.8	48.2	0.0	95.71	4.29	0.0	100.0	0.0	0.0
SA 1	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
SA 2	75.87	24.13	0.0	75.87	24.13	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Israel	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0

Table 6.5: Local results London Target - Timeframe = 7 days, MaxOccur = 50

Threshold	100%			60%			30%			25%		
Source	D	U	F	D	U	F	D	U	F	D	U	F
Iceland	13.37	86.63	0.0	17.69	82.31	0.0	17.69	82.31	0.0	17.69	82.31	0.0
Amsterdam	17.5	82.5	0.0	17.5	82.5	0.0	17.5	82.5	0.0	17.5	82.5	0.0
Ger2	17.64	82.36	0.0	17.64	82.36	0.0	17.64	82.36	0.0	17.64	82.36	0.0
Sweden	8.98	91.02	0.0	17.64	82.36	0.0	17.64	82.36	0.0	17.64	82.36	0.0
Milan	13.36	86.64	0.0	15.81	84.19	0.0	17.71	82.29	0.0	17.71	82.29	0.0
Chicago2	4.79	95.21	0.0	13.49	86.51	0.0	13.9	86.1	0.0	17.84	82.16	0.0
LA2	0.33	99.67	0.0	4.74	95.26	0.0	9.13	90.87	0.0	12.17	87.83	0.0
Chile	0.95	99.05	0.0	1.53	98.47	0.85	2.91	97.09	1.35	10.9	89.1	1.39
SA 1	4.72	95.28	0.0	8.99	91.01	0.0	13.3	86.7	0.0	13.3	86.7	0.0
SA 2	4.77	95.23	0.0	9.06	90.94	0.0	9.06	90.94	0.0	13.45	86.55	0.0
Israel	6.29	93.71	0.0	7.41	92.59	0.0	8.11	91.89	0.0	8.11	91.89	0.0
Threshold	20%			15%			10%			5%		
Source	D	U	F	D	U	F	D	U	F	D	U	F
Iceland	17.69	82.31	0.0	19.33	80.67	0.0	19.33	80.67	0.0	21.74	78.26	0.19
Amsterdam	17.5	82.5	0.0	17.5	82.5	0.0	19.09	80.91	0.0	19.12	80.88	1.53
Ger2	17.64	82.36	0.0	20.07	79.93	0.0	20.07	79.93	0.0	20.28	79.72	0.0
Sweden	17.64	82.36	0.0	17.64	82.36	0.0	17.64	82.36	0.0	18.44	81.56	0.0
Milan	17.71	82.29	0.0	17.71	82.29	0.0	17.71	82.29	0.0	18.35	81.65	0.52
Chicago2	17.84	82.16	0.0	17.84	82.16	0.0	17.84	82.16	0.0	17.84	82.16	0.0
LA2	13.44	86.56	0.0	13.44	86.56	0.0	13.44	86.56	0.0	13.44	86.56	0.0
Chile	10.95	89.05	1.39	11.56	88.44	1.54	19.08	80.92	1.6	19.76	80.24	1.6
SA 1	17.59	82.41	0.0	17.59	82.41	0.0	17.59	82.41	0.0	19.21	80.79	0.0
SA 2	13.45	86.55	0.0	13.45	86.55	0.0	17.81	82.19	0.0	22.77	77.23	0.0
Israel	8.11	91.89	0.0	8.14	91.86	0.0	14.13	85.87	0.0	16.67	83.33	0.0

Table 6.6: Local results London Target - Timeframe = 0.1 days, MaxOccur = 5

Entering an anomaly state requires K consecutive values above the defined threshold; to leave an anomalous state, it requires the same number of consecutive values below that threshold. In some instances, due to the RTT values via some relays being borderline anomalous but with some values above or below that threshold, the state of anomaly is never entered, because there are not enough consecutive values to trigger the event, and the same happens when trying to 'leave' the anomaly state. This parameter (*maxOccur*) does not only influence the time it takes to detect an anomaly (since K consecutive values are required) but how often a probe enters, and, leaves an anomalous state. It can be seen that a higher value of K provides a slightly better detection accuracy (Table 6.4 and Table 6.5) but at the cost of a higher detection time (takes longer to detect the anomaly). Another parameter that affects the detection of anomalies is *Timeframe*. As was previously mentioned, the detection module requires data to calculate what is considered the average 'normal' value as well as the new data to compare against that value.

The *Timeframe* is then the total amount of time (and respective data) available to calculate the average, disregarding any previously identified as anomalous values unless it cannot calculate the average due to a low number of data points, in which case it will use any data within that time frame. With this being the case, if the module cannot calculate an average value without being forced to use anomalous values, the average value may take values close to those of anomalous instances, which will lead to more anomalies being undetected. This can be seen when comparing Table 6.4 and Table 6.6. For the same *Threshold* and *maxOccur* values, the value of undetected anomalies increases with the lower *Timeframe* as can be seen, for example, for *Threshold* of 5% in the Iceland probe for which the value of undetected anomalies is 0% when using a *Timeframe* of 7 days and 78.26% when using a *Timeframe* of 0.1 days. Ideally, this value should be reasonably big but, from the results obtained, 7 days is already a good amount of information to calculate the average values since it allows for a relatively accurate result.

Threshold	100%			60%			30%			25%		
Source	D	U	F	D	U	F	D	U	F	D	U	F
Iceland	75.33	24.67	0.05	99.91	0.09	0.17	100.0	0.0	0.5	100.0	0.0	0.59
Amsterdam	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.03	100.0	0.0	0.26
Ger2	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Sweden	50.03	49.97	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Milan	75.02	24.98	0.0	97.52	2.48	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Chicago2	25.19	74.81	0.0	74.84	25.16	0.0	95.36	4.64	0.0	100.0	0.0	0.0
LA2	0.05	99.95	0.0	25.04	74.96	0.0	50.55	49.45	0.0	69.91	30.09	0.0
Chile	1.7	98.3	0.92	3.4	96.6	1.51	11.82	88.18	1.8	31.09	68.91	1.87
SA 1	25.02	74.98	0.0	50.01	49.99	0.0	75.0	25.0	0.0	75.05	24.95	0.0
SA 2	21.57	78.43	0.0	50.03	49.97	0.0	51.82	48.18	0.0	74.74	25.26	0.0
Israel	50.04	49.96	0.0	75.02	24.98	0.0	100.0	0.0	0.02	100.0	0.0	0.02
Threshold	20%			15%			10%			5%		
Source	D	U	F	D	U	F	D	U	F	D	U	F
Iceland	100.0	0.0	0.73	100.0	0.0	0.92	100.0	0.0	1.33	100.0	0.0	3.03
Amsterdam	100.0	0.0	2.19	100.0	0.0	7.9	100.0	0.0	39.72	100.0	0.0	72.52
Ger2	100.0	0.0	0.02	100.0	0.0	0.16	100.0	0.0	0.87	100.0	0.0	3.54
Sweden	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.03	100.0	0.0	2.67
Milan	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.09	100.0	0.0	14.37
Chicago2	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.1	100.0	0.0	2.2
LA2	75.01	24.99	0.0	75.01	24.99	0.0	75.01	24.99	0.02	75.03	24.97	0.41
Chile	49.69	50.31	1.91	61.97	38.03	1.99	96.39	3.61	2.15	100.0	0.0	2.41
SA 1	99.98	0.02	0.0	99.98	0.02	0.0	99.98	0.02	0.0	99.98	0.02	0.0
SA 2	75.02	24.98	0.0	75.54	24.46	0.02	100.0	0.0	0.02	100.0	0.0	0.03
Israel	100.0	0.0	0.02	100.0	0.0	0.02	100.0	0.0	0.42	100.0	0.0	3.56

Table 6.7: Local results London Target - Timeframe = 7 days, MaxOccur = 1

This parameter has a more notorious impact in the detection of anomalies than *maxOccur* as can be seen in Table 6.4 and Table 6.7 if one examines the value of undetected anomalies; it does, however, have an influence on the value of false detections as well.

Threshold	100%			60%			30%			25%		
Source	D	U	F	D	U	F	D	U	F	D	U	F
Iceland	0.33	99.67	0.0	25.25	74.75	0.0	75.08	24.92	0.0	75.47	24.53	0.0
Amsterdam	25.25	74.75	0.0	25.25	74.75	0.0	75.08	24.92	0.0	75.1	24.9	0.0
Ger2	25.25	74.75	0.0	25.25	74.75	0.0	75.08	24.92	0.0	75.08	24.92	0.0
Sweden	1.72	98.28	0.0	25.25	74.75	0.0	50.16	49.84	0.0	50.16	49.84	0.0
Milan	0.33	99.67	0.0	25.25	74.75	0.0	72.32	27.68	0.0	75.09	24.91	0.0
Chicago2	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
LA2	75.08	24.92	0.0	75.08	24.92	0.0	75.08	24.92	0.0	75.08	24.92	0.0
Chile	2.04	97.96	1.35	28.79	71.21	1.51	54.72	45.28	1.87	66.41	33.59	1.89
SA 1	0.36	99.64	0.0	0.36	99.64	0.0	25.24	74.76	0.0	32.5	67.5	0.0
SA 2	0.35	99.65	0.0	0.35	99.65	0.0	26.73	73.27	0.0	47.15	52.85	0.0
Israel	0.33	99.67	0.0	25.25	74.75	0.0	51.26	48.74	0.0	75.09	24.91	0.0
Threshold	20%			15%			10%			5%		
Source	D	U	F	D	U	F	D	U	F	D	U	F
Iceland	77.91	22.09	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.1
Amsterdam	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Ger2	77.51	22.49	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Sweden	52.99	47.01	0.0	97.6	2.4	0.0	100.0	0.0	0.0	100.0	0.0	0.0
Milan	75.4	24.6	0.0	75.47	24.53	0.0	75.47	24.53	0.0	100.0	0.0	0.0
Chicago2	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0	100.0	0.0	0.0
LA2	75.08	24.92	0.0	75.08	24.92	0.0	75.08	24.92	0.0	75.08	24.92	0.0
Chile	77.52	22.48	1.89	77.78	22.22	1.91	95.71	4.29	1.93	99.74	0.26	1.93
SA 1	75.08	24.92	0.0	75.1	24.9	0.0	100.0	0.0	0.0	100.0	0.0	0.0
SA 2 0.0		73.0	27.0	0.0	75.1	24.9	0.0	100.0	0.0	0.0	100.0	0.0
Israel	75.09	24.91	0.0	76.49	23.51	0.0	100.0	0.0	0.0	100.0	0.0	0.0

Table 6.8: Local results Chicago 1 Target - Timeframe = 7 days, MaxOccur = 5

The other parameter that is present in the tables is *maxOccur*. As previously explained, it defines the number of consecutive occurrences needed to enter or exit an anomalous state. Let us examine how exactly it affects the detection of BGP routing anomalies. Table 6.7 and Table 6.5 are two extreme examples of the importance of this parameter. The first table seems to be the perfect scenario: barely any false alarms or undetected anomalies. However, since it takes 50 consecutive values to trigger an anomaly state (and return to its normal state), depending on the time interval these values are obtained, it might be too late when an anomaly is detected since it can already be happening for $50 \cdot \mathbf{T}$ time instances. The second table will detect an anomaly with only 1 value above the threshold but that also means it will trigger an anomalous state even when no ongoing anomaly is occurring (as can be seen by the false detections value in the table formerly mentioned) since all it takes is a network or hardware delay to happen once. This parameter then defines both the time it takes to detect an anomaly (or exit an anomalous state), and also takes part in the accuracy of such detections. A higher value will make the detection happen slower but with more accuracy, and a lower value will make the detection happen faster but at the potential cost of accuracy. These parameters will then be up to the end user to define, but the ones that were found most optimised regarding the detection of global anomalies will be presented. Now that the effect of these parameters on the detection of local anomalies has been explained, it is now needed to evaluate the impact they have on the detection at a global scale.

6.2.2 Global detection

From what can be seen in tables Table 6.5, Table 6.6 and Table 6.7, a low *Timeframe* has a negative effect on the detection, and, as such, the higher value of 7 will be used for the global detections; a *maxOccur* of only 1 leads to a higher value of false alarms, so it will also be discarded; lastly, a *maxOccur* of 50 would take too long to detect an anomaly since the data is collected once per minute, so it will also be discarded. The differences in detection by using *maxOccur* values of 10 and 5 are negligible, so a value of 5 for *maxOccur* was picked, since that value will take half the time to detect anomalies compared to the higher value of 10. The *Threshold* values of 100% and 60% have, as can be seen on the previously shown tables, a high value of undetected anomalies and the values 5% and 10% have a high value of false detections, so these values were not considered when analysing the global detection. Taking into account the number of active probes used to gather the data that was analysed, a value of 40% for ρ (global threshold, which means that a global anomaly will start after 40% of the active probes monitoring a target enter an anomaly state, as previously explained in Chapter 3) was chosen. Let us take a look at the London target first with a local *threshold* value of 30%.

	Detected			
Source	LA 1	Madrid	Moscow	SP 1
Iceland	Yes	Yes	Yes	Yes
Amsterdam	Yes	Yes	Yes	Yes
Germany 2	Yes	Yes	Yes	Yes
Sweden	Yes	Yes	Yes	Yes
Milan	Yes	Yes	Yes	Yes
Chicago 2	Yes	Yes	Yes	Yes
LA 2	No	No	Yes	Yes
Chile	No	No	No	No
SA 1	Yes	Yes	Yes	Yes
SA 2	Yes	No	No	Yes
Israel	Yes	Yes	Yes	Yes
%	81.8	63.6	81.8	90.9

Table 6.9: Global detection results for London Target - Timeframe = 7 days, MaxOccur = 5, Threshold = 30%

From what can be seen in Table 6.9, almost all probes except Chile and LA 2 detected the anomaly via LA1, which can be explained due to the relatively high RTT from Chile to London and that the network from Chile to LA1 to London may yield better RTT times than Chile directly to London (different transatlantic cables - Figure 6.8). In LA 2, it is due to the proximity of the anomaly, since it is within the same geographical area. When traffic was being redirected through Madrid, Chile, LA 2, SA 1 and SA 2 were unable to detect the anomaly. For LA 2 and Chile it can be explained due to the relatively high RTT to London, and since the deviation from London to Madrid is not that high, it goes unnoticed by both probes. SA 1 and SA2 traffic to Europe has several submarine cables through which it could pass directly to Portugal, Spain, Italy, France and other European countries. In any of those cases, the deviation via Madrid is not noticeable since it is not big enough (RTT wise) compared to the normal values to trigger a detection. For the traffic being redirected through Moscow, Chile and SA 1 probes did not detect the anomaly, since the RTT increment is not significant enough compared to the normal RTT values observed on both probes. The last

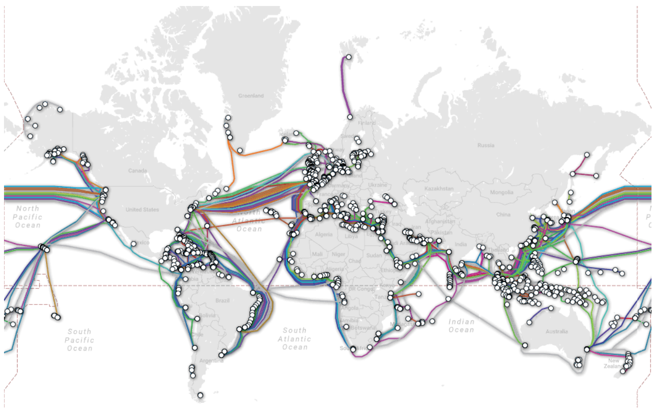


Figure 6.8: Submarine network cables.
www.submarinecablemap.com

deviation was via SP1, and all probes except Chile detected it. Since the deviation is relatively close to Chile's probe and likely sharing the same transatlantic cables as SP1 to reach Europe, it was unnoticed. Even with some probes not being able to detect the anomalies, the global anomalies regarding the London target can be detected as long as the value of the global threshold ρ is below 63.6%.

Detected				
Source	LA 1	Madrid	Moscow	SP 1
Iceland	Yes	No	Yes	Yes
Amsterdam	Yes	Yes	Yes	Yes
Germany 2	Yes	Yes	Yes	Yes
Sweden	Yes	Yes	Yes	Yes
Milan	Yes	Yes	Yes	Yes
Chicago 2	No	Yes	Yes	Yes
LA 2	No	No	No	Yes
Chile	Yes	No	No	No
SA 1	Yes	No	No	Yes
SA 2	Yes	No	No	Yes
Israel	Yes	Yes	Yes	Yes
%	90.9	45.5	63.6	90.9

Table 6.10: Global detection results for Ger1 Target - Timeframe = 7 days, MaxOccur = 5, Threshold = 30%

For the Ger1 target, the parameter values used were the same as the ones for the London target. From Table 6.10 it is possible to see that only the LA 2 probe did not detect the anomaly via LA 1, due to it being too close to the probe, while the remaining probes detected it. For the anomaly via Madrid, only 45.5% of the probes detected the anomaly. The reason for Chile's failure to detect it is the same as for the target in London. As for Chicago 2's failure to detect the anomaly, it as to do with the RTT from Ger1 to Madrid being small compared to the RTT value from Chicago 2 to Ger1, and, as such it did not go over the local threshold value. The same reason applies for LA 2 and Iceland. In the case of SA 1 and SA 2, it is the same reason as the one given for the London target. Looking at the deviation via Moscow, it is possible to see that 63.6% of the probes detected the anomaly, with Chile, LA2, SA 1 and SA 2 being unable to do so. The high RTT normal values from all these probes to the target combined with a comparatively low RTT value from Ger 1 to Moscow, make it so that the deviation does not cause the RTT values to go beyond the local threshold value. Lastly, the deviation via SP1 was detected by 90.9% of the probes with only Chile's probe not detecting the anomaly, due to the same reason as the one given for the London target. The global anomalies regarding the Ger1 target can be detected as long as the value of the global threshold is below 45.5%.

	Detected			
Source	LA 1	Madrid	Moscow	SP 1
Iceland	No	No	No	Yes
Amsterdam	No	No	No	Yes
Germany 2	No	No	No	Yes
Sweden	No	No	No	Yes
Milan	No	No	No	Yes
Chicago 2	No	Yes	Yes	Yes
LA 2	No	Yes	Yes	Yes
Chile	No	No	No	No
SA 1	No	No	No	No
SA 2	No	No	No	No
Israel	No	No	No	Yes
%	0	18.2	18.2	63.6

Table 6.11: Global detection results for Hong Kong Target - Timeframe = 7 days, MaxOccur = 5, Threshold = 30%

Regarding the Hong Kong target, it is possible to see that the deviation via LA 1 was not detected by any of the probes (Table 6.11). For the European probes, the reason lies in the RTT value increment from the 'normal' path (through Russia) to the abnormal path (through the transatlantic cables, LA 1, and then the submarine cables from the USA to Hong Kong) not being high enough to go beyond the threshold. The same reasoning applies to Chile's, SA 1's, and SA 2's probes. Regarding the deviation via Madrid, only 18.2% of the probes detected the anomaly. The reason that Chicago 2 and LA 2 detected the anomaly lies with the path from these probes to Hong Kong under normal circumstances being faster, RTT wise, than going through Europe. The same probes that detected the former anomaly also detected the one via Moscow, for the same reason; the European probes, however, do not detect it since Moscow is nearby the normal path taken from those countries to Hong Kong, and SA 1 and SA 2 do not detect it due to the same reasoning as for the deviation via LA 2. On the other hand, the deviation via SP1 is detected by 72.7% of the probes monitoring the target with the exception of Chile, which can be explained by the anomaly happening near the normal path of the traffic to the target, and SA 1 and SA 2, which can be explained by the relatively high RTT value to Hong Kong under normal conditions, and that the delay added by going through SP1 is not enough to go beyond the local threshold. In this instance, not all the global anomalies could be detected, since only the anomaly via SP1 is detected by over 40% of the probes.

Detected				
Source	LA 1	Madrid	Moscow	SP 1
Iceland	Yes	No	Yes	Yes
Amsterdam	Yes	No	Yes	Yes
Germany 2	Yes	No	Yes	Yes
Sweden	Yes	No	No	Yes
Milan	Yes	No	Yes	Yes
Chicago 2	Yes	Yes	Yes	Yes
LA 2	No	Yes	Yes	Yes
Chile	No	Yes	Yes	No
SA 1	No	No	No	Yes
SA 2	No	No	No	Yes
Israel	Yes	No	No	Yes
%	54.5	27.3	63.6	81.8

Table 6.12: Global detection results for Chicago 1 Target - Timeframe = 7 days, MaxOccur = 5, Threshold = 30%

Lastly, for the Chicago 1 target (Table 6.12), the deviation via LA 1 was detected by 54.5% of the probes, not being detected by Chile, LA 2, SA 1, and SA 2. For LA 2 the reason is simple, the anomaly is too close, RTT wise, to the target for it to be detected. For Chile, SA 1, and SA 2, the anomaly does not represent a significant increase in the RTT value that would go beyond the local threshold, since the RTT value from these probes to Chicago 1 is relatively high, compared to the RTT from Chicago 1 to LA 2. For the Madrid deviation, only 27.3% of the probes detected the anomaly. The reason for the failure to detect the anomaly lies with the relatively high RTT value from European and Asian countries to Chicago 1 compared with the RTT value from Chicago 1 to LA 2. Regarding the deviation via Moscow, 63.6% of the probes detected the anomaly. As for the deviation via SP1, 81.8% of the probes detected the anomaly. The global anomalies regarding the Chicago 1 target can be detected as long as the value of the global threshold is below 27.3%.

Although Chile detected a false anomaly on 05/31, it was the only probe to do so, and, as such, there were no global false anomalies detected. From these tables it is possible to see that the local threshold value of 30%, with a global threshold value of 40%, can detect 12 out of 16 anomalies (4 for each target), which is a global detection rate of 75%. Since the number of global false detections is 0% with the local threshold of 30% (for a global threshold of 40%), the data was then analysed with lower local threshold values, to find the most accurate one that does not raise false detections.

Detected				
Source	LA 1	Madrid	Moscow	SP 1
Iceland	Yes	Yes	Yes	Yes
Amsterdam	Yes	Yes	Yes	Yes
Germany 2	Yes	Yes	Yes	Yes
Sweden	Yes	Yes	Yes	Yes
Milan	Yes	Yes	Yes	Yes
Chicago 2	Yes	Yes	Yes	Yes
LA 2	No	Yes	Yes	Yes
Chile	Yes	No	Yes	No
SA 1	Yes	Yes	Yes	Yes
SA 2	Yes	No	Yes	Yes
Israel	Yes	Yes	Yes	Yes
%	90.9	81.8	100	90.9

Table 6.13: Global detection results for London Target - Timeframe = 7 days, MaxOccur = 5, Threshold = 15%

From the tested values of local threshold, the value that detects the most global anomalies while not triggering false global anomalies (for any global threshold greater or equal to 30%) is 15%. Let us start by comparing Table 6.9 and Table 6.13. The percentage of probes detecting the anomalies increased from 81.8, 63.6, 81.8, 90.9 to 90.9, 81.8, 100, 90.9 (via LA1, Madrid, Moscow and SP1 respectively) while having a 0% of false global anomalies, which is a clear improvement.

	Detected			
Source	LA 1	Madrid	Moscow	SP 1
Iceland	Yes	Yes	Yes	Yes
Amsterdam	Yes	Yes	Yes	Yes
Germany 2	Yes	Yes	Yes	Yes
Sweden	Yes	Yes	Yes	Yes
Milan	Yes	Yes	Yes	Yes
Chicago 2	Yes	No	Yes	Yes
LA 2	No	Yes	Yes	Yes
Chile	Yes	No	Yes	Yes
SA 1	Yes	No	Yes	Yes
SA 2	Yes	No	Yes	Yes
Israel	Yes	Yes	Yes	Yes
%	90.9	63.6	100	100

Table 6.14: Global detection results for Ger1 Target - Timeframe = 7 days, MaxOccur = 5, Threshold = 15%

For the Ger1 target, analysing Table 6.10 and Table 6.14 shows that the percentage of probes detecting the anomalies increased from 90.9, 45.4, 63.6, 90.9 to 90.9, 63.6, 100, 100 (via LA1, Madrid, Moscow and SP1 respectively), with 0% of false global anomalies, which is also an improvement.

	Detected			
Source	LA 1	Madrid	Moscow	SP 1
Iceland	No	No	No	Yes
Amsterdam	No	No	No	Yes
Germany 2	No	No	No	Yes
Sweden	Yes	Yes	No	Yes
Milan	Yes	Yes	Yes	Yes
Chicago 2	No	Yes	Yes	Yes
LA 2	No	Yes	Yes	Yes
Chile	No	Yes	Yes	No
SA 1	No	No	No	Yes
SA 2	No	No	No	Yes
Israel	Yes	Yes	Yes	Yes
%	27.3	54.5	45.4	90.9

Table 6.15: Global detection results for Hong Kong Target - Timeframe = 7 days, MaxOccur = 5, Threshold = 15%

For the next target, Hong Kong, from the tables Table 6.11 and Table 6.15 it is possible to see that the percentage values increased from 0, 18.1, 18.1, 72.7 to 27.3, 54.5, 45.4, 90.9 (via LA1, Madrid, Moscow and SP1 respectively), with some probes detecting false local anomalies, but without those being enough to trigger a false global anomaly; this is also an improvement.

	Detected			
Source	LA 1	Madrid	Moscow	SP 1
Iceland	Yes	Yes	Yes	Yes
Amsterdam	Yes	Yes	Yes	Yes
Germany 2	Yes	Yes	Yes	Yes
Sweden	Yes	Yes	Yes	Yes
Milan	Yes	No	Yes	Yes
Chicago 2	Yes	Yes	Yes	Yes
LA 2	No	Yes	Yes	Yes
Chile	Yes	Yes	Yes	Yes
SA 1	Yes	No	Yes	Yes
SA 2	Yes	No	Yes	Yes
Israel	Yes	No	Yes	Yes
%	90.9	63.6	100	100

Table 6.16: Global detection results for Chicago 1 Target - Timeframe = 7 days, MaxOccur = 5, Threshold = 15%

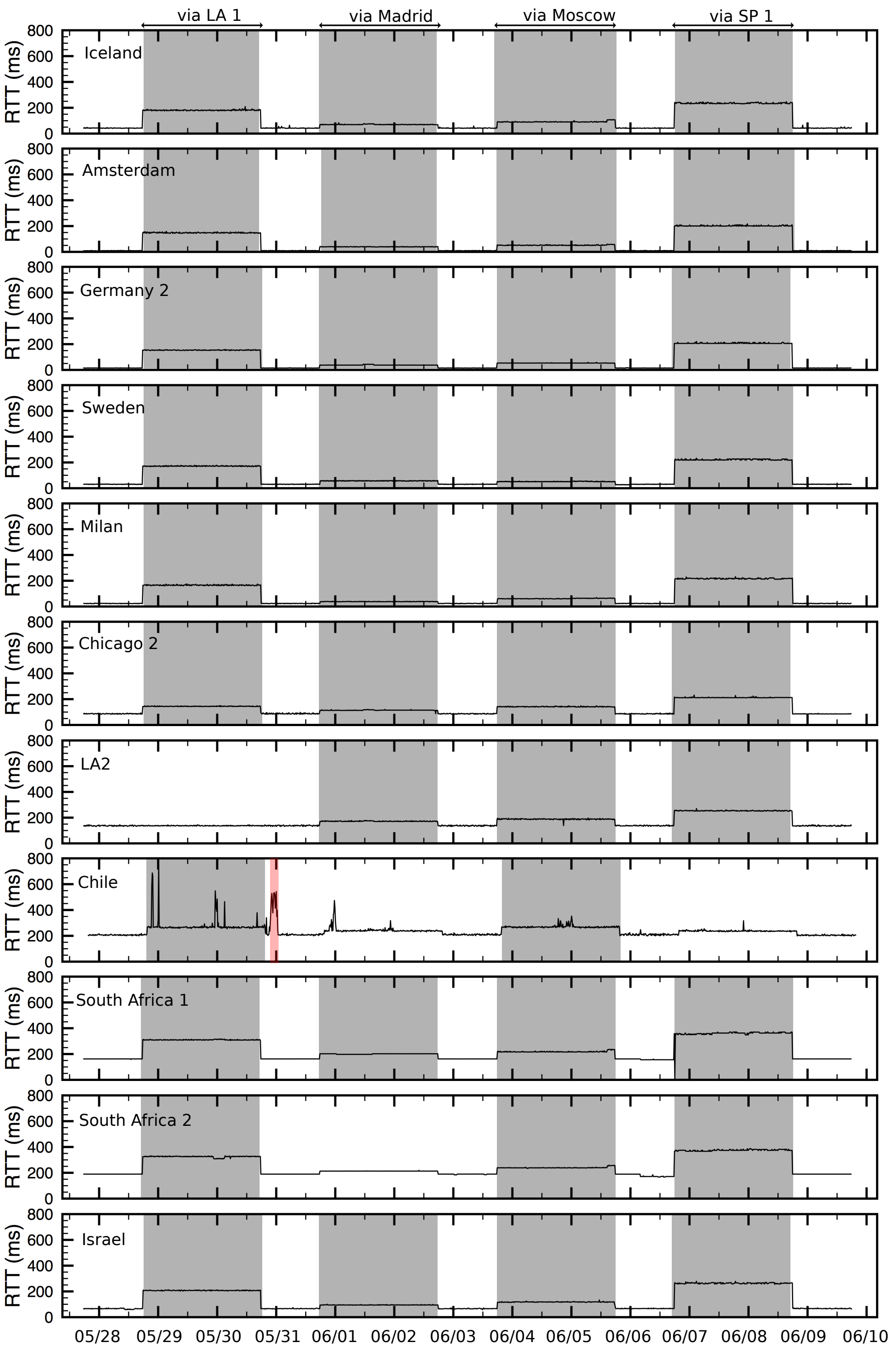


Figure 6.9: Detection results - London target, Timeframe = 7 days, Local Threshold = 15%

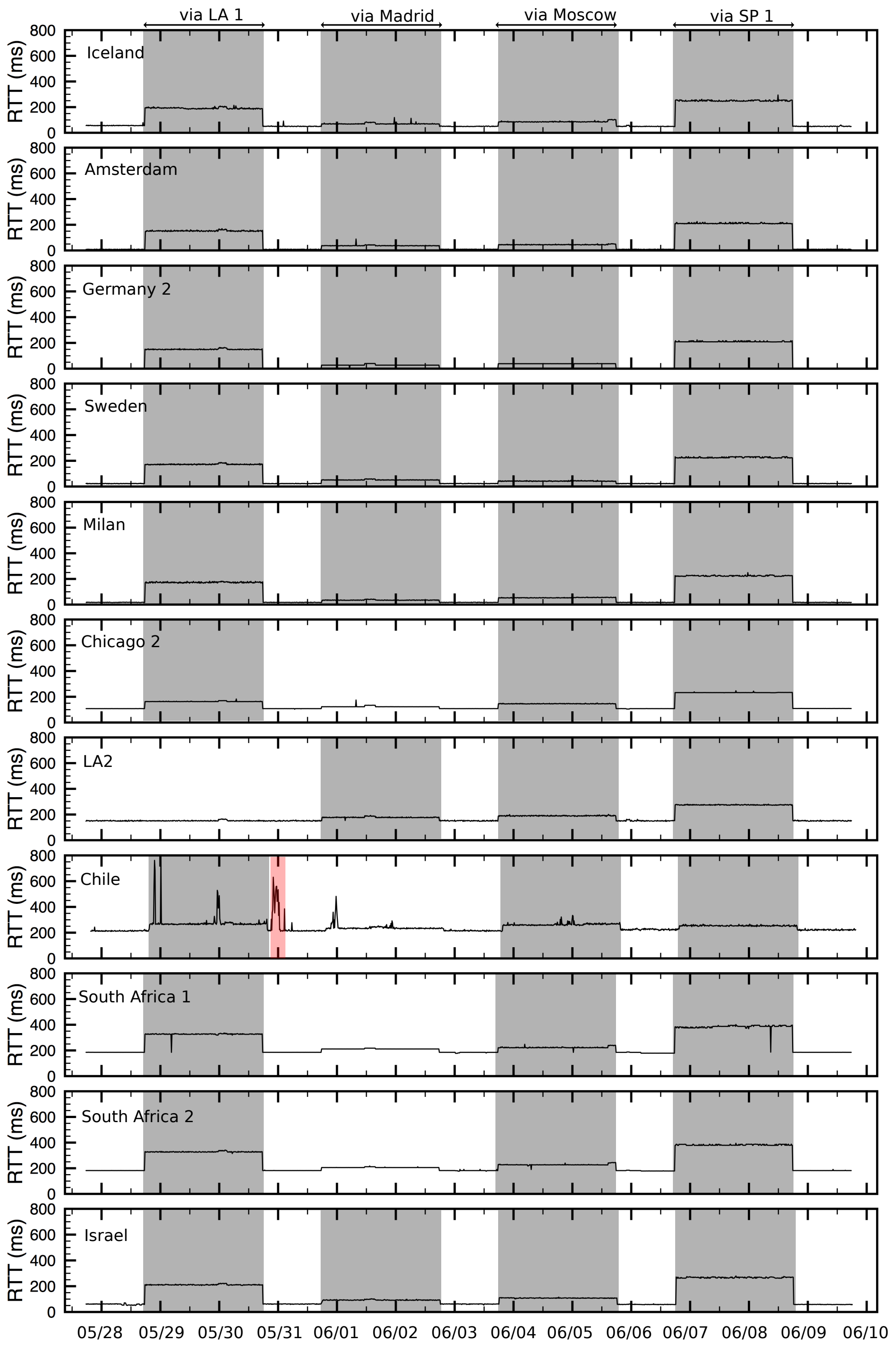


Figure 6.10: Detection results - Ger 1 target, Timeframe = 7 days, Local Threshold = 15%

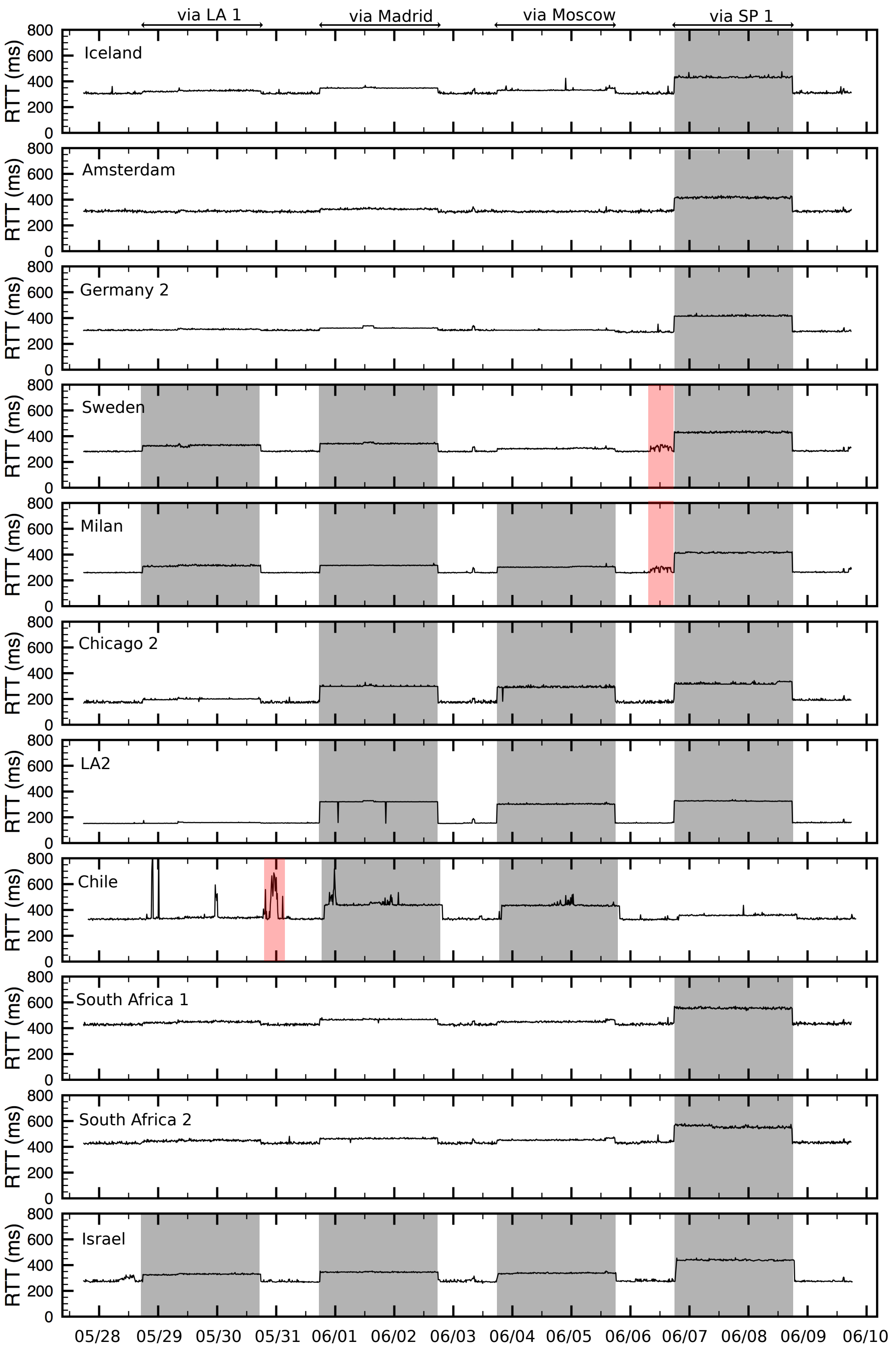


Figure 6.11: Detection results - Hong Kong target, Timeframe = 7 days, Local Threshold = 15%

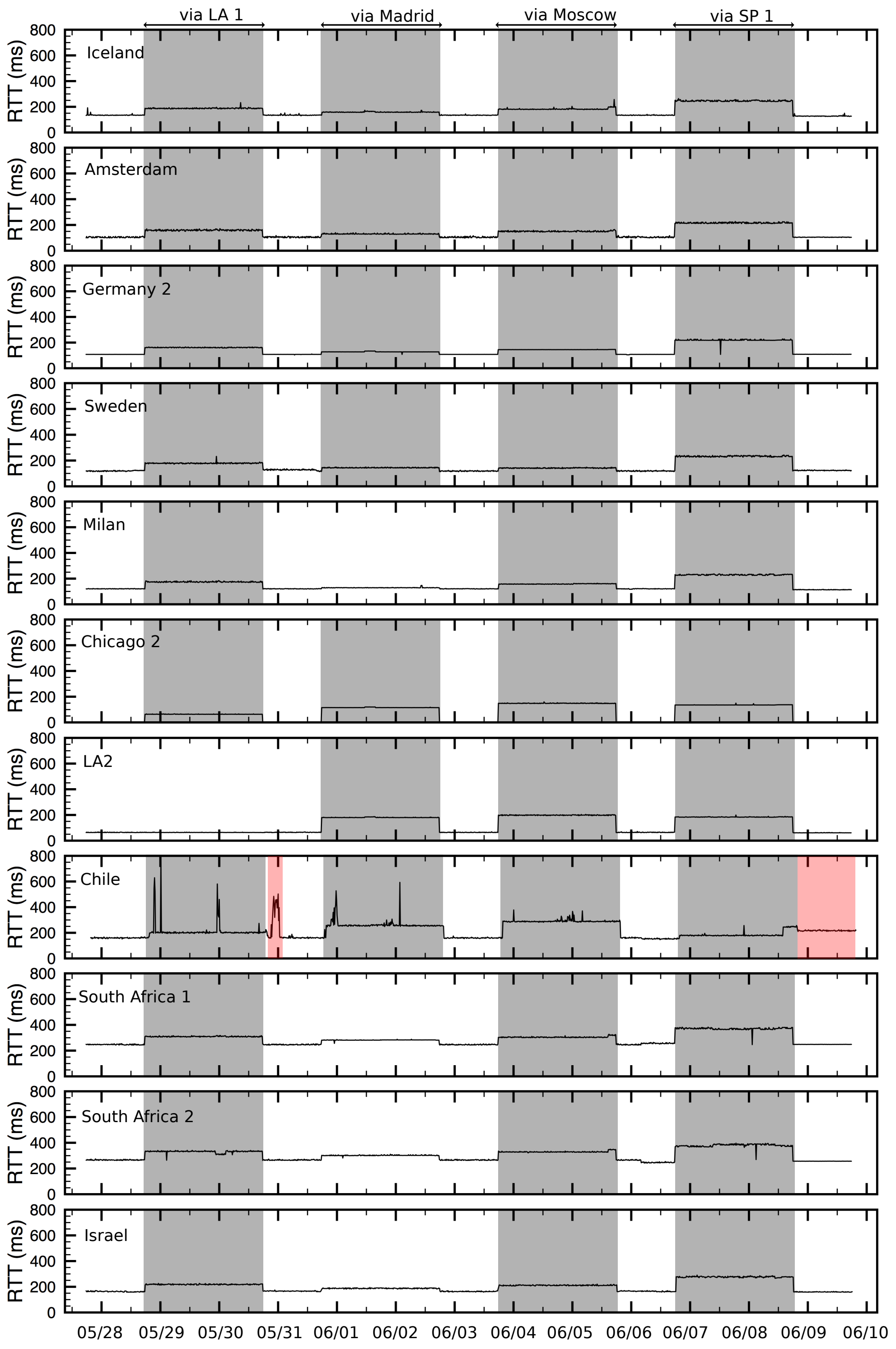


Figure 6.12: Detection results - Chicago 1 target, Timeframe = 7 days, Local Threshold = 15%

Lastly, for the Chicago 1 target, from tables Table 6.12 and Table 6.16 it is possible to see that the percentage values increased from 54.5, 27.3, 63.6, 81.8 to 90.9, 63.6, 100, 100 (via LA1, Madrid, Moscow and SP1 respectively). It is then clear to see that this local threshold yields better detection rates than the former. There is, however, a catch: if the global threshold value is low, it may lead to false global anomalies. This can be solved by either using a higher global threshold (such as 40%) or by deploying more probes which will help offset false detections. The stability of the probes is also important since, the more stable the probe, the fewer the occurrences of false local anomalies, and a more accurate detection rate. However improved the values were, the global detection rate is still not 100%. With this local threshold and a global threshold of 30%, the global detection rate is 15 out of 16 (4 for each target) which represents 93.75% of the global anomalies. The global detection of the anomaly via LA 1 for the Hong Kong target is still low (27.3%) which could be improved by deploying probes near that target. Figure 6.9, Figure 6.10, Figure 6.11 and Figure 6.12 show a visual representation of the data obtained, the anomalies detected (in grey), and false detections (red). For the London target, it is possible to see that the Iceland, Amsterdam, Germany 2, Sweden, Milan, Chicago 2, Chile, South Africa 1, South Africa 2 and Israel probes detect the anomaly via LA 1. Iceland, Amsterdam, Germany 2, Sweden, Milan, Chicago 2, LA 2, South Africa 1, South Africa 2, Israel probes detect the anomaly via Madrid. All the probes detect the anomaly via Moscow. All probes except Chile detect the anomaly via SP 1. Regarding the Ger 1 (Germany 1) target, it is possible to see that all probes except LA 2 detect the anomaly via LA 1; Iceland, Amsterdam, Germany 2, Sweden, Milan, LA 2 and Israel probes detect the anomaly via Madrid. All probes detect the anomaly via Moscow, and via SP 1. For the Hong Kong target, Sweden, Milan and Israel probes detect the anomaly via LA 1. Sweden, Milan, Chicago 2, LA 2, Chile and Israel detect the anomaly via Madrid. Milan, Chicago 2, LA 2, Chile and Israel detect the anomaly via Moscow, and all but Chile detect the anomaly via SP 1. It is also possible to see in this figure that there are 3 local false detections, 2 of them occurring simultaneously in Sweden's and Milan's probes. If the global threshold were set to be less than 19%, the global false detection would increase.

6.3 Methodology 2 results

The data necessary for validating this methodology has been obtained using the same probes previously mentioned. The setup was done differently due to constraints explained in 6.1. The probes configured as sources remain the same as for the previous methodology, but the probes that served as relay were also used as sources. All old sources took measurements to all the targets plus the relays, and the relays also took measurements to the targets. The data was then explored and used to validate the methodology. The data was gathered from 24th of October, 2017 to 1st of November, 2017.

Since the first part of the methodology proposed in 3.2.2 detects anomalies using the path vector, when it was being deviated through all the several relays without simulating the TTL adjustment that can be performed to hide the extra hops, all anomalies were detected. To determine if this methodology is able to detect attacks that are perfectly hidden by attacking the right AS and adjusting the TTL value, i.e., such as the path under attack is exactly the same as when it is not under attack, there was a need to simulate this attack. This attack, which can be see in Figure 3.4, is an extreme case of a BGP routing attack. It is extreme because the conditions required to be able to do it are not easy to obtain. The attacker must know the path of the traffic from multiple networks to the intended victim's network and be able to hijack the right router in one of the ASes in the path. To simulate this, the scenario depicted in Figure 6.5 was used. In order to achieve that scenario, it was required that one of the nodes in the normal path could be used as the hijacked router. To do so, the following 4 different routes were considered as the normal path: (i) Amsterdam to Chicago 1 via Madrid; (ii) Chicago 2 to Hong Kong via Madrid; (iii) Milan to Hong Kong via Madrid, and (iv) LA 2 to London via Madrid. For these, an attack that would deviate the traffic was simulated. This simulation made the traffic go through another VPS from the middle node that was used as a hijacked router (Madrid). The deviations were through: São Paulo 1, Moscow, Moscow and Iceland, respectively. For all of these, the normal path was kept intact, so only the second part of the methodology is used to check for anomalies. The threshold value used is 15% since that was the value which was found optimal for methodology 1.

Starting with (i), its normal path is composed of the following ASNs and respective RTT values in milliseconds: 43350 (0.29 ms), 1200 (3.05 ms), 8220 (31.66 ms), 39020 (31.77 ms), 61440 (32.06 ms), 39020 (32.599 ms), 174 (142.304 ms). With the generated anomaly the path is the same, but the RTT values are 0.30 (ms), 3.06 (ms), 31.71 (ms), 31.90 (ms), 256.07 (ms) and 366.31 (ms). The anomaly was detected on ASN 61440.

For (ii), the normal path, and respective RTT value, is: 39020 (0.01 ms), 61440 (1.09 ms), 174 (110.24 ms), 57169 (112.71 ms), 39020 (162.32 ms), 174 (272.27 ms), 9304 (431.97 ms), 9293 (452.56 ms). With the generated anomaly the path is the same, but the RTT values are 0.1 (ms), 1.09 (ms), 110.26 (ms), 189.32 (ms), 240.72 (ms), 350.30 (ms), 508.56 (ms) and 509.97 (ms). The anomaly was detected on ASN 57169. This is one case where the anomaly would go undetected by the first methodology since the RTT from source to destination is 452.56 ms under normal circumstances and 509.97 ms under attack which would be undetected with a threshold of 15%, $452.56 \cdot 1.15 = 520.44$, which is higher than the RTT value when under attack, making it go undetected.
Regarding (iii), the normal path is: 39020 (0.02 ms), 61440 (0.24 ms), 33182 (0.70 ms), 2914 (0.80 ms), 174 (172.53 ms), 39326 (173.54 ms), 39020 (176.47 ms), 174 (197.51 ms), 2914 (197.97 ms). With the generated anomaly, the RTT values are 0.02 (ms), 0.25 (ms), 0.72 (ms), 0.83 (ms), 174.85 (ms), 238.34 (ms), 241.78 (ms), 262.62 (ms) and 266 (ms). The anomaly was detected on ASN 39326.

For the last scenario, (iv), the path is: 20836 (0.32 ms), 8220 (24.41 ms), 39020 (24.65 ms), 57169 (77.02 ms), 174 (186.57 ms), 9304 (356.28 ms), 9293 (374.86 ms). With the generated anomaly, the RTT values are 0.32 (ms), 24.43 (ms), 24.78 (ms), 155.02 (ms), 264.48 (ms), 420.20 (ms) and 422.86 (ms). This scenario would also be undetected with the first anomaly with a threshold of 15% since the RTT value from source to destination under attack is lower than the average value multiplied by the threshold. All anomalies were then detected.

As for the global detection of the anomalies, this methodology proved to be able to detect all anomalies that were studied in this dissertation, with an accuracy of 100% when the attack was not hidden. It is needed to note that the number of false alarms could potentially be higher with this methodology. Although changes in the path, AS wise, are not too common, they are not too rare either. This issue could be mitigated by having a larger timeframe of the routing information from the probes to the targets. This methodology could be adapted to use a variable path vector when analysing the data. What this means is that the new path vector could possibly be different by N ASes without triggering an anomaly. This case was not considered during the testing of this methodology, but could be considered in the future.

6.4 Platform performance

This section will display the performance results for both the CPM and Mainframe. For the CPM, the hardware specs of the VPS used are: Intel XEON E5645 with a clock speed of 2.4GHz (1 core), 512MB RAM, running Ubuntu 16.04. The type and speed of the RAM are unknown. Mainframe specs are: 6 cores with a clock speed of 3.0GHz, 8GB of RAM, running Ubuntu 16.04.

Type	Avg CPU %	Avg RAM (MB)	Max CPU %	Max RAM (MB)
Standalone	0.24	23.09	36.38	23.3
W/ Modules	0.26	36.6	38.86	36.96

Table 6.17: Performance results of CPM.

Table 6.17 contains the test results of the CPM, which is used in the Probes. As can be seen, it is very lightweight, resource wise. An average CPU usage of 0.24% without any modules, and 0.26% with the two modules presented in this dissertation. Maximum CPU usage at 36.38% and 38.86%, with and without modules, respectively. The usage of RAM is also quite low, which was one of the goals, since VPSes with high amounts of RAM are far more expensive than the ones with low amounts, such as 128MB. At 23.09MB and 36.6MB of RAM usage on average (with and without modules, respectively), it is quite light. Regarding the Mainframe, the performance of this component was tested with, and without, probes and analysis modules. The countermeasures modules were left from these tests since they are not evoked on a time cycle basis. The results are as follow.

Type	Avg CPU %	Avg RAM (MB)	Max CPU %	Max RAM
Standalone	0.07	53.02	15.50	55.20
W/ Modules & 10 Probes	0.10	68.88	22.25	72.20

Table 6.18: Performance results of Mainframe.

From Table 6.18 it is possible to see that the Mainframe is also lightweight. With an average CPU usage of 0.07% without modules or probes, and 0.10% with two analysis modules as well as 10 probes. Maximum CPU usage at 15.50% and 22.25%, respectively. The usage of RAM is also quite low, at an average of 53.02MB and 68.88MB of RAM, respectively. These results are based on the state of the machine running the Mainframe as a whole, not the isolated Python code.

6.5 Result comparison

The results previously discussed regarding the first methodology show that it is possible to detect global BGP routing anomalies with a detection rate of 93.75% using a low local threshold (15%) combined with a relatively low global threshold as well (40%) without triggering false detections. To improve the detection rate and avoid false detections, the number of probes should be increased and more probes deployed near some of the targets which have no nearby probes (such as Hong Kong). The second methodology demonstrated an accuracy rate of 100% when the attack was not done under perfect conditions, while also detecting some anomalies that would go (locally) undetected by the first methodology.

Chapter 7

Conclusion and Future Work

This dissertation proposed the development and implementation of a modular platform capable of detecting BGP routing attacks. The modularity was achieved by giving the platform the ability to receive and execute new pieces of code, as long as that code is written in Python, or Python wrapped. To aid that modularity, the platform is also able of generating code regarding ORM databases, to permit the insertion and retrieval of data from the database by the added modules. The ability to add, remove and edit probes as well as the ability to add, remove and edit targets and respective data gathering modules with ease is also a factor in the overall usage and viability of the platform. Together with all the other features, such as different access levels for users, the ability to activate countermeasures on given targets, automatic management of the probes in case they become unresponsive and the ability to detect the anomalies form, altogether, a useful platform.

This platform would not be able to detect any anomalies without the ability to obtain and analyse data. With that in mind, four modules were created: two for data gathering and two for data analysis. These were created to implement the two methodologies proposed in this dissertation. With these methodologies implemented, the platform is then able to detect the anomalies. With the first methodology, the accuracy rate for the tested data set is of 93.75%, while for the second methodology, the accuracy detection rate with the first part of the methodology is 100% for both local and global anomalies, with the second part having an accuracy detection rate of 100% for local anomalies.

Another goal was to create a platform that is low on hardware requirements. That was accomplished by using a lightweight framework, in this case Python Flask. For the probes, the average usage of CPU and RAM was 0.26% and 36.6MB, respectively. For the Mainframe, the average usage of CPU and RAM was 0.10% and 68.88MB, respectively.

The goals of this dissertation were then met, by providing a modular, lightweight platform capable of detecting BGP routing anomalies, using two different methodologies. There is, however, still room for improvement. Such improvements may be:

- Improvement of the Graphical User Interface.
- New modules (and methodologies).

- New features, such as the ability to send email notifications to users.
- Ability to use different parameter values for the data gathering modules per probe and or target.
- Create automatic setups for different WSGIs.
- Create indexes for faster queries (database).
- More extensive testing of methodology 2.

Appendices

Appendix A

Mainframe installation guide and usage

A.1 Setup

To install the mainframe, there are a few requirements that have to be met. The machine must have Python3, pip3, MongoDB and Python3 virtual env installed. With those requirements, the setup is simple. Inside the .zip containing the Mainframe's code, there is a Python script, *setupMainframe.py*, a folder, *RSAgen* and a .zip file, *mainframe.zip*. Other files and folders are present, but they are not important for the installation of the Mainframe. Simply run the *setupMainframe.py* file using the command *python3 setupMainframe.py*. The script will then unzip the file, create the structure that is necessary for the mainframe to function and generate the RSA key pairs in the required folder. Inside the new folder (*mainframe*), all the necessary files are found. If the user wishes to change the certificate (which is advisable), it must be placed inside the folder named *certs*, on the root path of the Mainframe (*./mainframe/certs/*).
Before running the Mainframe, two files must be configure: *flask_configs.txt* and *mainframe.json*. A new user also needs to be created. It can be created by using the *manageUsers.py* script located on the folder *user_scripts*. After configuring those two files and creating the user, all that is required is to start the platform, with the command *venv/bin/python mainframe.py*.
To access the interface, navigate to the url *http(s)://ipAddress:port/adminControl*. From there, all the options available on the platform can be seen. To manage probes, click the Probes tab. The same for users, alarms, countermeasures, logs and modules. The setup of probes is done automatically by the Mainframe when the setup option is pressed, as long as the probe in question meets the requirements.

A.2 Altering docker image

Inside the .zip file containing the platform, there is a folder named *docker_image* which contains everything that is on the current docker image residing on DockerHub. Changes can be made freely, as long as the Python code remains the same as well as the needed requirements. The Python code may be altered, but depending on the changes, those may have to be reflected on parts of the code of the Mainframe. If the user wishes to use another docker image, the scripts located on *mainframe/probe_scripts/* will need to be altered to reflect the changes. Keep in mind that if changes are made after some probes have already been initiated, those will be running the older version of the image, and, as such, may need to be updated either manually or by removing them from the Mainframe and setting them up again. For more information on how to perform changes to a docker image, refer to this : `https://docs.docker.com/get-started/`

A.3 GUI In-Depth Guide

When accessing the platform's website, the first page seen is the **Login** page. In there, the email address and password must be inserted to gain access to the remaining pages.
After logging in, the user will be redirected to the **Overview** page, where some information is displayed. Disk and CPU usage of the Mainframe, and the percentage of probes in the platform that are currently online.
Next up is the **Probes** page. This page is composed of two tabs: *Probes* and *Targets*. On the first one, it is possible to see all the probes currently in the system as well as their information and status (online, offline, not setup). Clicking on an entry to the probe's table will show multiple actions, depending on the status of the probe. If it is online, the option to restart or stop the probe is present. If it is offline, the start option is given. If it has yet to be setup, that option will show, together with two fields for the username and password of a user with root privileges on the intended probe. Note that this password is not stored anywhere, and will be only used for the initial setup. Together with all these options is the ability to delete a probe from the platform. To add a new probe, simply click on the *Add Probe* button on the top right corner. A modal will appear where the user can enter the information relative to the probe and add it to the system. After adding a probe, clicking that probe on the probe's table will bring up the option to initiate it.
On the Targets tab, it is possible to select a probe and visualise the targets it is monitoring, as well as the modules running for each target. To add a new target, simply select the probe and a *New Target* button will appear on the top right corner, where the user can then add the new target. To add, remove or change the modules running on a probe to any given target, simply click the target and select the desired modules.
The next page is the **Alarms** page, where a user can visualise the alarms that have happened, which contains information regarding the target, probe(s), description and the date of when it was raised.
The following page, **Users**, allows for the visualisation of all the users in the system, their name, email, role and last login. It is also possible to add a new user by clicking the *Add User* button and filling out the required details.
Next up is the **Modules** page, which displays two tabs: *Data Gathering* and *Analysis*. In each of the tabs, it is possible to add a new module, as well as visualise the existing ones and their attributes. To add a new module, click the *Upload Module* and fill the fields. For the data gathering modules, a name, a .zip file and a JSON file are required. The .zip file must contain all needed Python code files, and if there are any dependencies, they must be at the root of the folder named as *requirements.txt*. The JSON file must obtain the structure described in this dissertation, in chapter 4. For the data analysing modules, the .zip file should contain all code required, and any dependencies must be in the root folder with the name

requirements.txt. These requirements files must obey pip format in order to be installed.
The next page is the **Countermeasures** page. It is divided in two tabs, *Targets* and *Countermeasures*. On the first tab, a list of all the targets is displayed, as well as the currently active countermeasures for those targets. To add, remove or modify the active countermeasures for a target, simply click the target in question and alter the desired countermeasures. On the second tab, it is possible to add a new countermeasure by clicking on the *New Countermeasure* button, on the top right corner. Two scripts can be uploaded; the first one is the one to be executed when an alarm is raised for a target with that countermeasure active and the second one (*Reset script*) is used after the target leaves an anomalous state. Only the first script is required, but it is advisable that the reset script be also uploaded so that the changes made upon an alarm can be reverted.
Lastly, the **Logs** page shows internal logs, such as probe status changes and local anomalies.

Bibliography

[1] International Telecommunication Union, "ICT Facts and Figures 2015." http://www.itu.int/net/pressoffice/press_releases/2015/17.aspx#.VWSF32Bjq-Q.

[2] International Telecommunication Union, "ICT Facts and Figures 2017." http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2017.pdf.

[3] A. Pilosov and T. Kapela, "Stealing The Internet - An Internet-Scale Man In The Middle Attack," in *DEFCON16*, August 2008.

[4] J. Cowie, "The New Threat: Targeted Internet Traffic Misdirection," *Dyn Blog*, November 2013. http://dyn.com/blog/mitm-internet-hijacking/.

[5] D. Madory, "On-going BGP Hijack Targets Palestinian ISP," *Dyn Blog*, January 2015. http://dyn.com/blog/going-bgp-attack-targets-palestinian-isp/.

[6] Dough Madory, "UK traffic diverted through Ukraine." https://dyn.com/blog/uk-traffic-diverted-ukraine/.

[7] Cisco, "The Zettabyte Era: Trends and Analysis." https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html.

[8] Cisco, "Protecting Border Gateway Protocol for the Enterprise." https://www.cisco.com/c/en/us/about/security-center/protecting-border-gateway-protocol.html.

[9] D. K. T. Bates, R. Chandra and Y. Rekhter, "Multiprotocol extensions for bgp-4," RFC 4760, RFC Editor, January 2007. http://www.rfc-editor.org/rfc/rfc4760.txt.

[10] R. Chandra, P. Traina, and T. Li, "Bgp communities attribute," RFC 1997, RFC Editor, August 1996.

[11] J. Hawkinson and T. Bates, "Guidelines for creation, selection, and registration of an autonomous system (as)," BCP 6, RFC Editor, March 1996.

[12] Q. Vohra and E. Chen, "Bgp support for four-octet autonomous system (as) number space," RFC 6793, RFC Editor, December 2012. http://www.rfc-editor.org/rfc/rfc6793.txt.

[13] G. Huston and G. Michaelson, "Textual representation of autonomous system (as) numbers," RFC 5396, RFC Editor, December 2008.

[14] Y. Rekhter, T. Li, and S. Hares, "A border gateway protocol 4 (bgp-4)," RFC 4271, RFC Editor, January 2006. http://www.rfc-editor.org/rfc/rfc4271.txt.

[15] E. Chen, "Route refresh capability for bgp-4," RFC 2918, RFC Editor, September 2000.

[16] J. Postel, "Transmission control protocol," STD 7, RFC Editor, September 1981. http://www.rfc-editor.org/rfc/rfc793.txt.

[17] Q. Vohra and E. Chen, "Bgp support for four-octet as number space," RFC 4893, RFC Editor, May 2007.

[18] Cisco, "Border Gateway Protocol (BGP)." https://www.cisco.com/cpress/cc/td/cpress/fund/ith2nd/it2435.htm#xtocid226038.

[19] P. Traina, D. McPherson, and J. Scudder, "Autonomous system confederations for bgp," RFC 3065, RFC Editor, February 2001.

[20] Cisco, "BGP Best Path Selection Algorithm." https://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/13753-25.html.

[21] Stephen T. Kent, BBN Technologies, "Securing the Border Gateway Protocol - The Internet Protocol Journal - Volume 6, Number 3." https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-25/securing-bgp-s-bgp.html.

[22] Russ White, Cisco Systems, "Securing BGP Through Secure Origin BGP - The Internet Protocol Journal - Volume 6, Number 3." https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-25/securing-bgp-sobgp.html.

[23] Geoff Huston, APNIC and Randy Bush, IIJ, "SecuringSecuring BGP - The Internet Protocol Journal, Volume 14, No. 2." https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-52/142-bgp.html.

[24] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile," RFC 5280, RFC Editor, May 2008. http://www.rfc-editor.org/rfc/rfc5280.txt.

[25] R. Bush and R. Austein, "The resource public key infrastructure (rpki) to router protocol," RFC 6810, RFC Editor, January 2013.

[26] M. Lepinski, S. Kent, and D. Kong, "A profile for route origin authorizations (roas)," RFC 6482, RFC Editor, February 2012.

[27] DYN, "Pakistan hijacks YouTube." https://dyn.com/blog/pakistan-hijacks-youtube-1/.

[28] D. Massey, D. Pei, Y. Wu, B. Zhang, and L. Zhang, "PHAS: A Prefix Hijack Alert System," http://irl.cs.ucla.edu/papers/originChange.pdf.

[29] BGPmon Network Solutions Inc, "BGPmon." https://bgpmon.net/.

[30] Renesys, “Routing Alarms®,” https://b2b.renesys.com/tech/datasheets/Renesys_Routing_Alarms_Datasheet.pdf.

[31] H. Yan, R. Oliveira, K. Burnett, D. Matthews, L. Zhang, and D. Massey, “BGPmon: A real-time, scalable, extensible monitoring system,” http://www.bgpmon.io/publications/catch09.pdf.

[32] U. Internet Research Lab, CS Department, “Cyclops.” https://cyclops.cs.ucla.edu/.

[33] H. Yan, D. Massey, E. Mccracken, and L. Wang, “BGPMon and NetViews: Real-Time BGP Monitoring System,”

[34] M. Cosovic, S. Obradovio, and L. Trajkovic, “Classifying anomalous events in BGP datasets,” in *2016 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, IEEE, may 2016. http://ieeexplore.ieee.org/document/7726739/.

[35] J. Schlamp, R. Holz, Q. Jacquemart, G. Carle, and E. W. Biersack, “HEAP: Reliable Assessment of BGP Hijacking Attacks,” *IEEE Journal on Selected Areas in Communications*, vol. 34, pp. 1849–1861, jun 2016. http://ieeexplore.ieee.org/document/7460217/.

[36] K. Balu, M. L. Pardal, and M. Correia, “DARSHANA: Detecting route hijacking for communication confidentiality,” in *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*, pp. 52–59, IEEE, oct 2016. http://ieeexplore.ieee.org/document/7778593/.

[37] *Investigating the nature of routing anomalies: Closing in on subprefix hijacking attacks*, 04/2015 2015. https://www.net.in.tum.de/fileadmin/bibtex/publications/papers/schlamp_TMA_1_2015.pdf.

[38] Q. Jacquemart, G. Urvoy-Keller, and E. Biersack, “A longitudinal study of bgp moas prefixes,” vol. 8406, pp. 127–138, 04 2014.

[39] J. L. d. S. Rosa, “Customer-side detection of bgp routing attacks,” 2016. http://hdl.handle.net/10773/17808.

[40] P. Salvador and A. Nogueira, “Customer-side detection of internet-scale traffic redirection,” in *16th International Telecommunications Network Strategy and Planning Symposium (NETWORKS 2014)*, September 2014.

[41] J. Sobrinho, “An algebraic theory of dynamic network routing,” *IEEE/ACM Transactions on Networking*, vol. 13, pp. 1160–1173, oct 2005.

[42] MongoDB, Inc., “MongoDB.” https://www.mongodb.com.

[43] R. G. Kefei Cheng, Meng Gao, “Analysis and research on https hijacking attacks,” College of Computer Science, Chongqing University of Posts and Telecommunications Chongqing.

[44] Armin Ronacher, “Flask Deployment.” http://flask.pocoo.org/docs/0.12/deploying/.

[45] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina, "Generic routing encapsulation (gre)," RFC 2784, RFC Editor, March 2000. http://www.rfc-editor.org/rfc/rfc2784.txt.